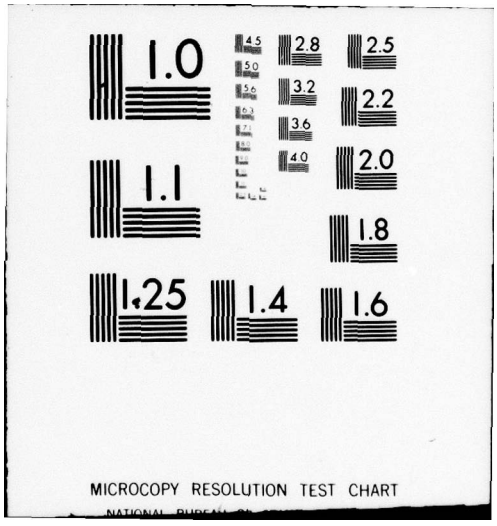
AD-A074 468 SRI INTERNATIONAL MENLO PARK CA F/G 9/2
ON-LINE PROGRAMMER'S MANAGEMENT SYSTEM.(U)
AUG 79 B L PARSLEY, H G LEHTMAN, S KAHN F30602-77-C-0185
UNCLASSIFIED RADC-TR-79-205 NL

END
DATE FILMED
11 -79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART

B.S.

LEVEL A

RADC-TR-79-205
Final Technical Report
August 1979

ON-LINE PROGRAMMER'S MANAGEMENT SYSTEM

Augmentation Resources Center

Bruce L. Parsley
Harvey G. Lehtman
Susan Kahn

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-205 has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

Add-1 495
Add-2 496

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-79-205	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): ON-LINE PROGRAMMER'S MANAGEMENT SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Sep 77 — Mar 79
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Bruce L. Parsley, Harvey G. Lehtman, Susan Kahn	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0185
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Augmentation Resources Center, 20705 Valley Green Drive, Cupertino CA 95014	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 55811803
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE), Griffiss AFB NY 13441	12. REPORT DATE: August 1979
	13. NUMBER OF PAGES: 72
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Same

18. SUPPLEMENTARY NOTES
RADC Project Engineer: Raymond A. Liuzzi (ISIE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Debugging	System Software
On-Line	JOVIAL
Software Engineering	Compilers
Programming Environments	
Computers	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This report is composed of studies that have been conducted to develop the NLS system as an on-line programming environment and to provide an on-line JOVIAL interactive debugger with the capabilities to debug JOVIAL language programs. The final report contains several design additions to the NLS system to create an on-line programming environment. A JOVIAL User's Guide prepared in Addendum Technical Report I provides an extensive set of commands for using the JDAD Debugger. Addendum Technical Report II provides a generalized approach to debugging and describes the NLS/NSW Do-All Debugger (DAD).

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410.281

EVALUATION

The work described in the final technical report and the addendum reports, one and two, represents a significant accomplishment in establishing parts of the framework of an on-line software programming environment.

The NLS system represents a significant programming tool that can be utilized to develop software programs. The high cost of developing software has been established in numerous studies. One way of reducing this cost is to develop standard programming environments. Parts of these programming environments will require sophisticated on-line debuggers. This effort establishes the feasibility of one such debugger for JOVIAL language programs.

The additions to NLS described in this effort also provide the types of capabilities that must be present in a standard software programming environment.

The result of this effort has been to extend the capability of tools needed in a software programming environment and establish procedures and methods for their implementation.

Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

On-Line Programmers Management System:
Final Technical Report
Augmentation Resources Center, Tymshare, Incorporated 1

Introduction 2

This report is submitted in fullfillment of Task/Technical Requirement 4.1.1 of the On-Line Programmers Management System project (RADC - SRI Contract Number F30602-77-C-0165; SRI - Tymshare Subcontract Number 14394). The Statement of Work said in part: 2a

TASKS AND TECHNICAL REQUIREMENTS 2a1

4.1.1 Develop a list of additions and modifications to the NLS family of tools to create an on-line programming environment based on the IBM evaluation of NLS's programming environment, the IBM Structured Programming Series, and the contractors own suggestions. 2a1a

4.1.2 Upon receipt of written approval from the contracting officer, or of his authorized representative, implement the approved suggestons from the list. 2a1b

4.1.3 Demonstrate a Jovial interactive debugger utilizing: 2a1c

4.1.3.1 Language module (LM) and operating system module (OS) for PDP-10 2a1d

4.1.3.2 Jovial compiler on PDP-10 2a1e

4.1.3.3 ARPA Network 2a1f

4.1.3.4 NLS/NSW Do-All Debugger (DAD) 2a1g

REPORTING REQUIREMENTS 2a2

-- An interim technical report describing the results of the software engineering tools study, including an ordered list of recommended additions to NLS. 2a2a

-- A final technical report. 2a2b

This is the report mentioned immediately above. 2b

Summary 3

As required by the Statement of Work, we have created a version of the ARC Do-All Debugger (DAD) for use with programs written in Jovial and which are compiled on the DEC PDP-10 running under the TENEX operating system. Attached is a user guide to this new Jovial DAD (JDAD). 3a

In the Interim Technical Report for this project submitted on 17 January 1979 as ARC Journal document (46236,), we presented an ordered list of additions and modifications to NLS programming tools. Since then we have worked on and completed the following of those recommendations: 3b

1. Enable NLS to start DAD and JDAD. 3b1

2. Encapsulate the JOVIAL compiler. 3b2

3. Document the encapsulation facility. 3b3

4. Generalize the PROGRAMS subsystem and its templates. 3b4

Additionally, we have created a detailed design for the following task: 3c

5. An interactive, conditional, iterative Process system. 3c1

The first four of the tasks have resulted in systems and tools which were demonstrated, along with the JDAD debugger, to the technical monitor of the project at ARC in Cupertino during the week of 25 April 1979. These tasks are discussed in the following sections of this report. The last section of this report is a glossary of terms that may be unfamiliar to readers. It is recommended that any reader unfamiliar with NLS or TENEX terminology read the glossary before reading the following sections. 3d

Detailed Discussion of Tasks Completed 4

Task 1: Enable NLS to start DAD and JDAD: A "Spliceable" DAD. 4a

Introduction 4a1

A facility was added permitting the DAD and JDAD debuggers to "splice" themselves into the fork structure for an already executing program. Before this feature was available, DAD or JDAD (which, for simplicity, will be called [J]DAD in the following) had to be started at the Exec level; from the debugger, it was then necessary to start the programs to be debugged. 4a1a

Under the new facility, a user would cause an executing program to be interrupted, e.g., via control-C, and issue a special command to the Exec. The command would result in [J]DAD being spliced into the proper place in the interrupted program's fork structure. 4a1b

As an example of the benefits of such a facility, consider a non-programmer doing normal work in some program. An unexpected bug may be encountered. That user may then call in a programmer who could access [J]DAD to investigate the bug with most of the bug's context intact. 4a1c

Tasks Performed to Create a Spliceable DAD or JDAD 4a2

The TENEX Exec was modified to create a new [J]DAD command which would splice in an instance of the [J]DAD program between the EXEC fork and its subsidiary forks (which presumably are running a program which one or more processes/forks.) After splicing in [J]DAD, execution would begin at a new entry point which would cause the new [J]DAD code described below to be executed. 4a2a

The [J]DAD Dispatch Module (which performs bookkeeping operations on the internal [J]DAD data structures concerned with the processes being debugged) and the [J]DAD Operating System Module (which interfaces [J]DAD to specific operating systems, in the current case TENEX and TOPS-20) were modified so [J]DAD could begin execution at an alternate entry point. If started at this entry point, [J]DAD would query the operating system for information concerning processes in the fork structure underneath it and the particular states of those forks. 4a2b

[J]DAD then establishes states of the forks and their

programs in its internal data structures comparable to the states which would exist if the program were initially executed under [J]DAD and the programmer had typed the control-L interrupt character to enter the debugger. 4a2c

After its intial polling of fork status information and state determination and establishment, [J]DAD may be continued in its usual fashion to set breakpoints, examine and change code, and in general perform its usual debugging tasks. 4a2d

Task 2: Encapsulate the JOVIAL compiler. 4b

Once the JOVIAL compiler is encapsulated, JOVIAL source code programs that are in NLS files may be directly compiled with a variation of the PROGRAMS subsystem's Compile File command, without the user directly having to go through intermediate steps involving a sequential file (as is presently the case). Also, the encapsulation of the JOVIAL compiler will allow the use of the LIBRARY subsystem for semi-automatic JOVIAL compilations. 4b1

Under the stategy implemented, source code for JOVIAL programs is composed and edited in AUGMENT. The "Compile JOVIAL" command is then invoked creating a temporary sequential file (invisible to the user). The encapsulated JOVIAL compiler is executed, using this sequential file as input, and also using switches which may be set as options to the command. Object code is compiled to the specified location; error messages and diagnostics are entered into the desired locations. 4b2

The following is the syntax for the command (extracted from the Command Meta Language grammar of the demonstration system): 4b3

```
"COMPILE"                                                    4b3a
   <"jovial program in file">                                4b3a1
   LSEL(#"OLDFILENAME") % the specification of the AUG-
   MENT JOVIAL source code file %                            4b3a2
   <"to">                                                    4b3a3
   LSEL(#"TEXT") % specification of the name of the ob-
   ject code file %                                          4b3a4
   [OPTION <"enter switches">                                4b3a5
      getswitches % any number of switches as outlined
      below may be specified %]                              4b3a5a
   [OPTION <"compool filename"> LSEL(#"TEXT")]               4b3a6
   CONFIRM                                                   4b3a7

The following are valid switches:                            4b3b

   ("CROSSREF"/ "ACROSS"/ "SYNTAX"/ "INDENT"/ "HBC"/
   "STATISTICS"/ "MACROCODE"/ "LOWSEG"/ "HISEG"/
   "NONEST"/ "ISD"/ "NOPT"/ "LISTCOPY"/ "NOSOURCE"/
   "DEFINE"/ "NOTRACE"/ "NOINFORM"/ "NOWARNING"/ "MIS-
   SION"/ "NOBACK"/ "HIGHSTAT"/ "KA10"/ "KI10"/ "MAGIC"/
   "ASSEM")                                                  4b3b1
```

Task 3: Document the encapsulation facility. 4c

Introduction 4c1

Encapsulation is a technique used to enable a process to control the execution environment of other processes. The controlling process does this by declaring that it wishes to trap selected system calls (JSYS's) when executed by other processes. When a monitored process attempts to execute a system call that will be trapped it is suspended and the monitoring process is notified. After gaining control the monitoring process may take any action it deems neccessary. It may handle the call itself or it may allow the monitored process to perform the call or it may modify the arguments to the system call before it is handled. 4c1a

This discussion here concerns taking a process which currently runs under TENEX or TOPS-20 and encapsulating it under AUGMENT. The purpose of encapsulation in this setting is to allow a user to run a process not currently provided directly in the AUGMENT environment, perhaps using AUGMENT files as input to the process and providing entry via AUGMENT-style commands. 4c1b

Description of Encapsulation Technique 4c2

There are three things that need to be done in AUGMENT in order to encapsulate a process designed to run under TENEX or TOPS-20: 4c2a

1. A correspondence table specifying each JSYS to be trapped and the address of the procedure which will handle the trapped JSYS must be included as a global declaration. The table below specifies that the three JSYS's BOUT, GTJFN, and PBOUT will be trapped by NLS and they will be handled by procedures named BOUTHNDLER, GTJFNHNDLER, and PBOUTHNDLER respectively. The table must end with -1, -1). 4c2a1

EXAMPLE: 4c2a2

```
% jsys handling correspondence table %                4c2a2a
   DECLARE                                            4c2a2a1
   jtraps = (                                         4c2a2a2
      $pbout, $pbouthndler,                           4c2a2a2a
      $bout, $bouthndler,                             4c2a2a2b
      $gtjfn, $gtjfnhndler,                           4c2a2a2c
```

-1, -1); 4c2a2a2d

2. Write code to start up the specified process in an inferior fork. 4c2a3

The steps to do this are: 4c2a3a

a.) First create the fork and load the process that is to run in the fork. This is done by first calling the backend procedure CRFORK with no arguments. CRFORK will return a fork handle number which must be saved for later procedure calls. The process is then loaded into the new fork's address space by a call on GTFKFILE. The arguments for GTFKFILE will be the fork handle number and the JFN of the save file to be loaded. 4c2a3a1

b.) Define which JSYS's are to be trapped and what procedure is to handle the jsys when the interrupt occurs. This is done by calling the backend procedure DEFINETRAPS with the address of the jsys correspondence table as its argument. which modifies two NLS tables: BITTABLE indicates which JSYS's are to be trapped by setting the appropriate bit and DTABLE indicates the address of the procedure to be called when a particular jsys is trapped. 4c2a3a2

c.) Call the backend procedure SETTRAPS with the fork handle of the process being encapsulated, a channel number over which interrupts will be transmitted to the controling fork, and the desired interrupt level as arguments. 4c2a3a3

d.) Save the terminal characteristics in case the encapsulated process changes them by calling the backend procedure SAVETRM. This procedure will return three results which must be saved for a later procedure call. The results are: the terminal mode word, terminal output control word one and terminal output control word two. 4c2a3a4

e.) Wait for the inferior fork to finish by calling the backend procedure WAITFRK with the fork handle as its argument. 4c2a3a5

f.) Call the backend procedure CLEANUP. This procedure will deactivate the channel connection, zero out BITTABLE and DTABLE, kill the inferior fork, and release the JFN of the sav file. 4c2a3a6

g.) Restore the terminal characteristics by calling the backend procedure RSTRTRM whose arguments will be the values returned by SAVTRM. 4c2a3a7

The procedure below is a typical example of how a process is encapsulated. All procedures mentioned can be found in the Encapsulator Module <nlsbesrc, encapsulator,>. 4c2a3b

EXAMPLE: 4c2a3c

```
(runit)   % run an encapsulated process %
PROCEDURE(savjfn) ;                                    4c2a3c1
   % Procedure description                             4c2a3c1a
      FUNCTION                                         4c2a3c1a1
         This procedure is responsible for
         starting up and running the
         encapsulated process as well as clean-
         ing things up when the process has ter-
         minated.  Note that three globals are
         required:                                     4c2a3c1a1a
            chan - is the channel used in
            issueing the interrupt usually a
            number in the high twenties or low
            thirties.                                  4c2a3c1a1a1
            ilev - interrupt it will usually be
            3.                                         4c2a3c1a1a2
            frkhandle - handle used to identify
            the encapsulated fork.                     4c2a3c1a1a3
      ARGUMENTS                                         4c2a3c1a2
         savjfn - jfn of sav file to be run in
         the new fork                                  4c2a3c1a2a
      RESULTS                                           4c2a3c1a3
         none                                          4c2a3c1a3a
      NON-STANDARD CONTROL                              4c2a3c1a4
         none                                          4c2a3c1a4a
      %                                                 4c2a3c1a5
   % Declarations %                                     4c2a3c1b
      (trmmode);                                        4c2a3c1b1
      (ctlwd1);                                         4c2a3c1b2
      (ctlwd2);                                         4c2a3c1b3
```

```
% define  channel and interrupt level for
jsys trapping in fork to be created %          4c2a3c1c
   chan _ 34;                                  4c2a3c1c1
   ilev _ 3;                                   4c2a3c1c2
% guard against errors %                       4c2a3c1d
   INVOKE(catch);                              4c2a3c1d1
% create new fork and enable all capabilities
%                                              4c2a3c1e
   frkhandle _ crfork();                       4c2a3c1e1
% get and load the .sav file to run in the
new fork %                                     4c2a3c1f
   gtfkfile(frkhandle, savjfn);                4c2a3c1f1
%  define jsys's to be trapped and set the
traps for the new fork %                       4c2a3c1g
   definetraps($jtraps);                       4c2a3c1g1
   settraps(frkhandle, chan, ilev);            4c2a3c1g2
% save terminal characteristics %              4c2a3c1h
   trmmode _ savtrm(:ctlwd1, ctlwd2);          4c2a3c1h1
% wait until it finishes %                     4c2a3c1i
   waitfrk(frkhandle);                         4c2a3c1i1
% cleanup %                                    4c2a3c1j
   cleanup(frkhandle, savjfn, chan);           4c2a3c1j1
% drop the catchphrase%                        4c2a3c1k
   DROP(catch);                                4c2a3c1k1
% Return %                                     4c2a3c1l
   RETURN;                                     4c2a3c1l1
% catchphrase definition %                     4c2a3c1m
   (catch) CATCHPHRASE();                      4c2a3c1m1
      BEGIN                                    4c2a3c1m1a
      CASE SIGNALTYPE OF                       4c2a3c1m1b
         = aborttype :                         4c2a3c1m1b1
            BEGIN                              4c2a3c1m1b1a
            DISABLE(catch);                    4c2a3c1m1b1b
            cleanup(frkhandle, savjfn, chan);
                                               4c2a3c1m1b1c
            END;                               4c2a3c1m1b1d
      ENDCASE;                                 4c2a3c1m1b2
      CONTINUE;                                4c2a3c1m1c
      END;                                     4c2a3c1m1d
END.                                           4c2a3c1n
```

There are several examples of processes which terminate only with a <^C> which would return control to the operating system rather than AUGMENT. One way to handle this is to have a handler detect when the desired processing has completed. The handler will then set a global flag and leave the encapsulated process frozen until such time that

the monitoring process sees the flag. Instead of waiting until the inferior fork finishes the monitoring procedure will have to look at the flag from time to time. When the flag is seen the monitoring routine can then kill the fork without creating any problems. 4c2a3d

3. Write the procedures that will handle the trapped jsys's. 4c2a4

The programmer will have to know something (perhaps, a great deal) about the process he wishes to encapsulate in order to write the handlers. Input and output are the primary concerns here. If the process expects a command from the terminal the handler may feed that command to the process. Also, an AUGMENT file may be handed to the process rather than a sequential file. The handler must know what the encapsulated process is requesting and how to satisfy that request. The handlers may have to keep track of many different states in order to do this successfully. 4c2a4a

JSYS handlers are passed four arguments: 4c2a4b

A fork handle 4c2a4b1

The current JSYS number 4c2a4b2

The address where the encapsulator saves the current fork PC 4c2a4b3

The address of a 16 word block containing the content of the fork's registers 4c2a4b4

JSYS handlers return two boolean values: 4c2a4c

handled 4c2a4c1

TRUE if JSYS has been handled 4c2a4c1a

FALSE let TENEX handle it 4c2a4c1b

unfreeze 4c2a4c2

TRUE to unfreeze the fork 4c2a4c2a

FALSE to leave it frozen % 4c2a4c2b

Encapsulator Module 4c3

The Encapsulator Module is a collection of AUGMENT routines located in <nlsbesrc,encapsulator,> and are used to handle the low level coding necessary to perform an encapsulation. 4c3a

It has three types of routines. One group of routines deals with starting up the encapsulation and cleaning up when it is finished: 4c3b

a.) Create the new fork. 4c3b1

b.) Define and set the traps. 4c3b2

c.) Reset global tables when encapsulation is finished.. 4c3b3

d.) Save and reset terminal characteristics. 4c3b4

Another group of routines are invoked by the interrupt mechanism. When an interrupt occurs DUMMYJSYSTRAPPSI is invoked and calls other routines in the Encapsulator Module to accomplish the following: 4c3c

a.) Save the state of the superior fork. 4c3c1

b.) Dispatch the interrupt to the appropriate handler. 4c3c2

c.) Restore the state of the superior fork. 4c3c3

d.) Let the inferior fork continue. 4c3c4

e.) Continue the superior fork in the state existing when the interrrupt occurred. 4c3c5

The remaining routines are designed to support writing in the inferior fork's accumulators and address space. 4c3d

Encapsulator Subsystem 4c4

The encapsulator subsystem is designed to assist the programmer faced with an encapsulation project as well as serving as a tool in discovering the nature of any process. It allows the user to run the process as if it were running uder the operating system while tracing all JSYS calls. 4c4a

The subsystem has a single command which asks the user to specify an executable program file (e.g., a TENEX SAV file) that is to be encapsulated, what information is desired (a trace of all JSYS calls, a frequency count, or both) and how that information is to be presented (displayed at the terminal or recorded in a file). The information obtained consists of the identification of the JSYS trapped and its arguments. Actually, the contents of the first four registers are obtained regardless of the number of arguments expected for the jsys. 4c4b

After the command is entered the specified process is encapsulated trapping each JSYS, obtaining the desired information about the JSYS and letting the operating system handle the call. Thus, the process will run as it would if it was running directly under the operating system. 4c4c

Currently, only JSYS's whose numbers are in the ranges 1 to 12B, 14B to 315B, and 317B to 337B are trapped. The TIME, GJINF, and JOBTM JSYS's are not included because TENEX does not execute them properly after they have been trapped. These JSYS's should never be trapped in any encapsulation. There may be others that cause problems and it would be useful if they are reported when encountered. 4c4d

If one is working in display mode and requests a trace, the information about each JSYS trapped will be displayed in the command feedback window and requires an OK before the process is continued. This allows the user to contemplate the flow of the process but can be very anoying if a single JSYS is executing inside a loop a large number of times. In typewriter mode no OK is required and the information will come out as fast as it is encountered. 4c4e

The frequency distribution table will not be displayed or recorded until the process terminates. Thus, one will not be able to get a frequency distribution of a process that is normally terminated by a <^C> since you will be returned to the operating system rather than the encapsulator subsystem. 4c4f

Task 4: Generalize the PROGRAMS subsystem and its templates. 4d

The AUGMENT PROGRAMS subsystem provides access to tools which aid in the program development process by permitting compilation and testing of programs and by permitting insertion of language dependent program entity templates. 4d1

The PROGRAMS subsystem is currently tailored for the L10 and CML languages. We would like to make its features available for other computer languages, e.g., JOVIAL. The main idea was to develop a framework and/or methodology for making the facilities of the PROGRAMS subsystem easily extensible to new computer languages. In the demonstration project, the commands and templates associated with the PROGRAMS system were expanded to deal with the JOVIAL, META (the ARC meta-compiler), and CML languages as well as L10. Structures relevant to these different languages were created as templates and entered as options into the PROGRAMS commands. 4d2

Appendix 1 outlines the command syntax for the new programs subsystem. Appendix 2 presents the current language dependent programming templates inserted via commands in the new PROGRAMS system as extracted from < SSSRC, PROGRAMS-TEMPLATES, >. 4d3

Task 5: An interactive, conditional, iterative Process system. 4e

NLS currently has a "Process commands" facility, but it is rather limited. Process commands are currently limited to be the equivalent of NLS commands -- it's roughly equivalent to what a user could type as input, e.g., there are currently no facilities for iteration, or for conditionally performing one set of commands rather than another set, or for interacting with the user in the midst of the execution of the Process commands. 4e1

The general idea would be to provide a "Process Language" that had all, or almost all, of the features of a computer language such as ALGOL, e.g., conditional statements, iterative statements, block structure, subroutines, constant and variable data declarations. In addition the language would have provisions for interactions with the user. Calls on L10 procedures probably would also be possible. 4e2

Process Language "programs" probably would be interpreted rather than compiled, probably in a manner similar to the present Process commands facility. 4e3

Some initial design thoughts on such a facility may be found in <29046,>. It was produced for the NSW project and appears as Appendix 3 to this report. It is applicable to the AUGMENT command language. Using this as a guide, we have created a design for the syntax for a complete Process Language. This syntax appears as Appendix 4. 4e4

Appendix 1: Command Syntax for the New AUGMENT PROGRAMS Subsystem 5

This appendix outlines the command syntax for the new PROGRAMS subsystem. The syntax is presented in a modified Command Meta Language description. 5a

Notes: 5a1

"!2" stands for "!2!" 5a1a

A final CONFIRM is left implicit for all commands. 5a1b

Curly-brackets are used to indicate a CML LOOP sort of thing; exit from the loop is made via an OK. 5a1c

Rules used elsewhere: 5a2

anyinput = anytext LSEL / fstructure <"at"> DSEL 5a2a

anytext = Character / Invisible / Text / Visible / Word / Statement 5a2a1

fstructure = 5a2a2

[OPTION <"Filtered: "> VWSPECS] (Branch / Group / Plex / Rest / File) 5a2a2a

userprog = Content-Analyzer / Sort-Key / Sequence-Generator!2 5a2b

compileunit = 5a2c

userprog / Run-Program / 5a2c1

Subsystem / 5a2c2

Grammar / Parse-Fe (Code/Data) / backend / Support-Module!2 / 5a2c2a

L10-Program / L1011-Program!2 / Cml-Program / Meta-Program / Jovial-Program / 5a2c3

Procedure / Coroutine!2 / Parse-Fe Function / Catchphrase!2 5a2c4

Insert % for L10 mode % 5a3

(Program 5a3a

/ compileunit 5a3b

/ If-Then-Else / Case!2 5a3c

/ For / Loop!2 / Do-Until / Do-While!2 / Until-Do / While-Do 5a3d

) 5a3e

<"to follow"> DSEL [LEVADJ] 5a3e1

```
Insert   % for CML mode %                                              5a4
   Program <"to follow"> DSEL [LEVADJ]                                 5a4a

Insert   % for Meta mode %                                             5a5
   Program <"to follow"> DSEL [LEVADJ]                                 5a5a

Insert   % for Jovial mode %                                           5a6
   ( Program                                                           5a6a
   / Compool!2  Directive / Compool!2 Source / Procedure               5a6b
   / If-Else / Switch                                                  5a6c
   / For By-While / For Then-While / While                             5a6d
   / Call / Procedure!2 Call Declaration                               5a6e
   )                                                                   5a6f
      <"to follow"> DSEL [LEVADJ]                                      5a6f1

Insert Comment <"to follow"> DSEL   % for any mode %                   5a7

Compile                                                                5a8
   ( Content-Analyzer <"in"> (anyinput / Program <"AT">
   DSEL)                                                               5a8a
   / Program <"AT"> DSEL <"using"> LSEL <"to"> LSEL                    5a8b
   / compileunit <"at"> DSEL                                           5a8c
   )                                                                   5a8d
      { Record <"errors"> <"at"> DSEL                                  5a8d1
      / Load  <"after compilation">                                    5a8d2
      / Filtered VWSPECS                                               5a8d3
      }                                                                5a8d4

Load         (Content-Analyzer / Program!2 / compileunit)
<"named"> LSEL                                                         5a9

Deinstitute  userprog                                                  5a10

Institute!2  userprog <"named"> LSEL                                   5a11

Run                                                                    5a12
   ( Run-Program <"named"> LSEL                                        5a12a
   / Process <"named"> LSEL                                            5a12b
      <"input from">                                                   5a12b1
         ( anyinput                                                    5a12b1a
         / Interactive <"with termination character"> LSEL             5a12b1b
         / Sequential-File <"named"> LSEL                              5a12b1c
         / No-Input                                                    5a12b1d
         )                                                             5a12b1e
      <"output to">                                                    5a12b2
         { Nls-File <"at"> LSEL                                        5a12b2a
         / Terminal                                                    5a12b2b
         / Sequential-File <"named"> LSEL                              5a12b2c
         / Registers ...                                               5a12b2d
```

```
        }                                                          5a12b2e
      <"wait for completion and then Kill?">                       5a12b3
        ( Yes / No <"notify at completion?"> (Yes/No) )            5a12b3a
      <"go?">                                                      5a12b4
   )                                                               5a12c

Kill          Process <"named"> LSEL                               5a13

Wait          <"for completion and then Kill"> Process
<"named"> LSEL                                                     5a14

Show          (Buffer / Insert-Mode / Process <"named"> LSEL)      5a15

Set                                                                5a16
   ( Buffer-Size <"to"> LSEL                                       5a16a
   / Insert-Mode <"to"> (Cml / Jovial / L10 / Meta)                5a16b
   )                                                               5a16c

Reset         (Buffer-Size / Insert-Mode)                          5a17

Delete!2      (All / Last)                                         5a18

Invoke!2      (Dad / Ddt!2)                                        5a19
```

```
Appendix 2:  Language Dependent Programming Templates                    6

   The following are the current programming templates inserted
   via commands in the new PROGRAMS system as extracted from <
   SSSRC, PROGRAMS-TEMPLATES, >:                                         6a

      (cml)                                                              6a1

         (grammar)                                                       6a1a
            %GR%   FILE SubSysName  % GRammar %                         6a1a1
               % COMPILE-INSTRUCTIONS %                                 6a1a1a
                  INCLUDE <SsSrc, NLS-Grammar, flags !subsystems>
                                                                        6a1a1a1
               % DECLARATIONS %                                         6a1a1b
                  INCLUDE <SsSrc, NLS-Grammar, declarations
                  !universal>                                           6a1a1b1
                  DECLARE COMMAND WORD                                  6a1a1b2
                     %subsystem command words (should be 100 to
                     127)%                                             6a1a1b2a
                     "COMMAND1" = 100,                                 6a1a1b2b
                     "COMMANDn" = 10n-1;                               6a1a1b2c
                     %%                                                6a1a1b2d
                  DECLARE FEFUNCTION                                    6a1a1b3
                     fefunc1,                                           6a1a1b3a
                     fefuncn;                                           6a1a1b3b
                  DECLARE FUNCTION                                      6a1a1b4
                     PROCESS = "PROCESSX" , PACKAGE = "PCKX" :          6a1a1b4a
                        xroutine1,                                      6a1a1b4a1
                        xroutinen;                                      6a1a1b4a2
                  DECLARE GLOBAL                                        6a1a1b5
                     global1,   % short description of global1 %
                                                                        6a1a1b5a
                     globaln;   % short description of globaln %
                                                                        6a1a1b5b
                     DECLARE VARIABLE                                   6a1a1b6
                        variable1,                                      6a1a1b6a
                        variablen;                                      6a1a1b6b
                        %%                                              6a1a1b6c
               % COMMON-RULES %                                         6a1a1c
                  INCLUDE <SsSrc, NLS-Grammar, rules !universal>        6a1a1c1
                                                                        6a1a1c1a
                  rule1 = <rule-body>;                                  6a1a1c2
                                                                        6a1a1c2a
                  rulen = <rule-body>;                                  6a1a1c3
                                                                        6a1a1c3a
                     %%                                                 6a1a1c3b
               % COMMANDS % SUBSYSTEM SubSysName KEYWORD
               "SUBSYSNAME"                                             6a1a1d
                  INITIALIZATION % for SubSysName %                     6a1a1d1
```

```
            initsubsys = xinitsubsys();                          6a1a1d1a
         TERMINATION % for SubSysName %                           6a1a1d2
            trmsubsys = xtrmsubsys();
                                                                 6a1a1d2a
         command1 COMMAND = "COMMANDWORD1"                        6a1a1d3
            <rule-body>                                          6a1a1d3a
            xroutine1(variable1, variable2);
                                                                 6a1a1d3b
         command2 COMMAND = "COMMANDWORD2"                        6a1a1d4
            <rule-body>                                          6a1a1d4a
            xroutine2(variable1, variable2);
                                                                 6a1a1d4b
         INCLUDE <SsSrc, NLS-Grammar, commands
         !universal>                                              6a1a1d5
         END.                                                     6a1a1d6
      FINISH                                                       6a1a1e

(program)                                                          6a1b
   %GR%   FILE SubSysName  % GRammar %                             6a1b1
      % COMPILE-INSTRUCTIONS %                                     6a1b1a
         INCLUDE <SsSrc, NLS-Grammar, flags !subsystems>
                                                                 6a1b1a1
      % DECLARATIONS %                                           6a1b1b
         INCLUDE <SsSrc, NLS-Grammar, declarations
         !universal>                                             6a1b1b1
         DECLARE COMMAND WORD                                    6a1b1b2
            %subsystem command words (should be 100 to
            127)%                                              6a1b1b2a
            "COMMAND1" = 100,                                  6a1b1b2b
            "COMMANDn" = 10n-1;                                6a1b1b2c
            %%                                                 6a1b1b2d
         DECLARE FEFUNCTION                                      6a1b1b3
            fefunc1,                                           6a1b1b3a
            fefuncn;                                           6a1b1b3b
         DECLARE FUNCTION                                        6a1b1b4
            PROCESS = "PROCESSX" , PACKAGE = "PCKX" :          6a1b1b4a
               xroutine1,                                     6a1b1b4a1
               xroutinen;                                     6a1b1b4a2
         DECLARE GLOBAL                                          6a1b1b5
            global1,   % short description of global1 %
                                                               6a1b1b5a
            globaln;   % short description of globaln %
                                                               6a1b1b5b
         DECLARE VARIABLE                                        6a1b1b6
            variable1,                                         6a1b1b6a
            variablen;                                         6a1b1b6b
            %%                                                 6a1b1b6c
      % COMMON-RULES %                                          6a1b1c
```

```
      INCLUDE <SsSrc, NLS-Grammar, rules !universal>
      rule1 = <rule-body>;
      rulen = <rule-body>;
         %%
   % COMMANDS % SUBSYSTEM SubSysName KEYWORD
   "SUBSYSNAME"
      INITIALIZATION % for SubSysName %
         initsubsys = xinitsubsys();
      TERMINATION % for SubSysName %
         trmsubsys = xtrmsubsys();
      command1 COMMAND = "COMMANDWORD1"
         <rule-body>
         xroutine1(variable1, variable2);
      command2 COMMAND = "COMMANDWORD2"
         <rule-body>
         xroutine2(variable1, variable2);
      INCLUDE <SsSrc, NLS-Grammar, commands
      !universal>
      END.
   FINISH

(110)

   (backend)
      %BE%   FILE SubSysName  % BackEnd %
         % DECLARATIONS %
            % Dispatch-Table: address of routine-name-
            string, address of routine %
               (subsysname) EXTERNAL = (
                  $"XROUTINE1", $xroutine1,
                  $"XROUTINEn", $xroutinen,
                  0,0);  % table must end with zero %
                  %%
            % Command Words %
               (cw1) CONSTANT = 100B;
               (cwn) CONSTANT = 10n-1B;
               %%
         % X-ROUTINES %
            (xroutine1)   % one-line description %
            (xroutinen)   % one-line description %
         % CORE-ROUTINES %
            (croutine1)   % one-line description %
```

6a1b1c1
6a1b1c1a
6a1b1c2
6a1b1c2a
6a1b1c3
6a1b1c3a
6a1b1c3b
6a1b1d
6a1b1d1
6a1b1d1a
6a1b1d2
6a1b1d2a
6a1b1d3
6a1b1d3a
6a1b1d3b
6a1b1d4
6a1b1d4a
6a1b1d4b
6a1b1d5
6a1b1d6
6a1b1e
6a2
6a2a
6a2a1
6a2a1a
6a2a1a1
6a2a1a1a
6a2a1a1a1
6a2a1a1a2
6a2a1a1a3
6a2a1a1a4
6a2a1a2
6a2a1a2a
6a2a1a2b
6a2a1a2c
6a2a1b
6a2a1b1
6a2a1b2
6a2a1c
6a2a1c1

```
      (croutinen)   % one-line description %          6a2a1c2
   % SUPPORT-ROUTINES %                               6a2a1d
      (sroutine1)   % one-line description %          6a2a1d1
      (sroutine1)   % one-line description %          6a2a1d2
   FINISH                                             6a2a1e

(case)                                                6a2b
   CASE case-exp OF                                   6a2b1
      cond-exp1 :                                     6a2b1a
         BEGIN                                        6a2b1a1
         END;                                         6a2b1a2
      ENDCASE;                                        6a2b1b

(catchphrase)                                         6a2c
   (catchname) CATCHPHRASE(arg1, arg2, arg3, arg4);   6a2c1
      BEGIN                                           6a2c1a
      CASE SIGNALTYPE OF                              6a2c1b
         = notetype :                                 6a2c1b1
         = helptype :                                 6a2c1b2
         = aborttype :                                6a2c1b3
            CASE SIGNAL OF                            6a2c1b3a
               = xxx :                                6a2c1b3a1
               = yyy :                                6a2c1b3a2
               ENDCASE;                               6a2c1b3a3
      ENDCASE;                                        6a2c1b4
      CONTINUE;                                       6a2c1c
      END;                                            6a2c1d

(content-analyzer)                                    6a2d
   (analyzer-name)   % CL: ;  one-line-description %
   PROCEDURE (arg1 <type>, ..., argn % => [meta-res] res1
   <type>, ..., resn %);                              6a2d1
      % Procedure description                         6a2d1a
         FUNCTION                                     6a2d1a1
            none                                      6a2d1a1a
         ARGUMENTS                                    6a2d1a2
            none                                      6a2d1a2a
         RESULTS                                      6a2d1a3
            none                                      6a2d1a3a
         NON-STANDARD CONTROL                         6a2d1a4
            none                                      6a2d1a4a
         %                                            6a2d1a5
      % Declarations %                                6a2d1b
      %procedure body%                                6a2d1c
      % Return %                                      6a2d1d
         RETURN;                                      6a2d1d1
      END.                                            6a2d1e
         %%                                           6a2d1e1
```

```
(coroutine)                                                    6a2e
   (coname)   % CL: ;  one-line-description %
   COROUTINE (arg1 <type>, ..., argn);                         6a2e1
      % Coroutine description                                  6a2e1a
         FUNCTION                                              6a2e1a1
            none                                               6a2e1a1a
         ARGUMENTS                                             6a2e1a2
            none                                               6a2e1a2a
         RESULT                                                6a2e1a3
            none                                               6a2e1a3a
         NON-STANDARD CONTROL                                  6a2e1a4
            none                                               6a2e1a4a
         %                                                     6a2e1a5
      % Declarations %                                         6a2e1b
      % Initial entry point %                                  6a2e1c
         PORT ENTRY                                            6a2e1c1
            % Initialization %                                 6a2e1c1a
         EXIT PCALL;                                           6a2e1c2
      %coroutine body%                                         6a2e1d
      END.                                                     6a2e1e
         %%                                                    6a2e1e1

(cml-program)                                                  6a2f
   %GR%   FILE SubSysName  % GRammar %                         6a2f1
      % COMPILE-INSTRUCTIONS %                                 6a2f1a
         INCLUDE <SsSrc, NLS-Grammar, flags !subsystems>
                                                               6a2f1a1
      % DECLARATIONS %                                        6a2f1b
         INCLUDE <SsSrc, NLS-Grammar, declarations
         !universal>                                           6a2f1b1
         DECLARE COMMAND WORD                                  6a2f1b2
            %subsystem command words (should be 100 to
            127)%                                              6a2f1b2a
            "COMMAND1" = 100,                                  6a2f1b2b
            "COMMANDn" = 10n-1;                                6a2f1b2c
            %%                                                 6a2f1b2d
         DECLARE FEFUNCTION                                    6a2f1b3
            fefunc1,                                           6a2f1b3a
            fefuncn;                                           6a2f1b3b
         DECLARE FUNCTION                                      6a2f1b4
            PROCESS = "PROCESSX" , PACKAGE = "PCKX" :          6a2f1b4a
               xroutine1,                                      6a2f1b4a1
               xroutinen;                                      6a2f1b4a2
         DECLARE GLOBAL                                        6a2f1b5
            global1,   % short description of global1 %
                                                               6a2f1b5a
            globaln;   % short description of globaln %
                                                               6a2f1b5b
         DECLARE VARIABLE                                      6a2f1b6
```

```
            variable1,                                          6a2f1b6a
            variablen;                                          6a2f1b6b
            %%                                                  6a2f1b6c
        % COMMON-RULES %                                        6a2f1c
            INCLUDE <SsSrc, NLS-Grammar, rules !universal>      6a2f1c1
                                                                6a2f1c1a
            rule1 = <rule-body>;                                6a2f1c2
                                                                6a2f1c2a
            rulen = <rule-body>;                                6a2f1c3
                                                                6a2f1c3a
            %%                                                  6a2f1c3b
        % COMMANDS % SUBSYSTEM SubSysName KEYWORD
        "SUBSYSNAME"                                            6a2f1d
            INITIALIZATION % for SubSysName %                   6a2f1d1
               initsubsys = xinitsubsys();
                                                                6a2f1d1a
            TERMINATION % for SubSysName %                      6a2f1d2
               trmsubsys = xtrmsubsys();
                                                                6a2f1d2a
            command1 COMMAND = "COMMANDWORD1"                   6a2f1d3
               <rule-body>                                      6a2f1d3a
               xroutine1(variable1, variable2);
                                                                6a2f1d3b
            command2 COMMAND = "COMMANDWORD2"                   6a2f1d4
               <rule-body>                                      6a2f1d4a
               xroutine2(variable1, variable2);
                                                                6a2f1d4b
            INCLUDE <SsSrc, NLS-Grammar, commands
            !universal>                                         6a2f1d5
            END.                                                6a2f1d6
        FINISH                                                  6a2f1e

    (do-until)                                                    6a2g
        DO                                                       6a2g1
            BEGIN                                               6a2g1a
            END                                                 6a2g1b
        UNTIL until-clause;                                      6a2g2

    (do-while)                                                    6a2h
        DO                                                       6a2h1
            BEGIN                                               6a2h1a
            END                                                 6a2h1b
        WHILE while-clause;                                      6a2h2

    (fefunction)                                                  6a2i
        (FEFunctionName)  % CL: ;  one-line description %
        PROCEDURE (reason, instruction, accumulator REF,
        argcount, arguments REF, saveword % => result %);       6a2i1
            % FEFunction description                           6a2i1a
```

```
      FUNCTION                                                  6a2i1a1
         none                                                  6a2i1a1a
      ARGUMENTS (show grammar arguments)                        6a2i1a2
         none                                                  6a2i1a2a
      RESULT (show grammar results)                             6a2i1a3
         none                                                  6a2i1a3a
      NON-STANDARD CONTROL                                      6a2i1a4
         none                                                  6a2i1a4a
      %                                                         6a2i1a5
   % Declarations %                                              6a2i1b
   CASE reason OF                                               6a2i1c
      = parsing: % being invoked for first time during
      command %                                                 6a2i1c1
         BEGIN                                                 6a2i1c1a
         % decide whether FF is on the correct path
         through the grammar %                                 6a2i1c1b
         IF % not on right path, may % THEN RETURN
         (notme);                                              6a2i1c1c
         % do processing %                                     6a2i1c1d
         saveword _ % word of context to be saved or 0
         - if 0, will not be called during backup,
         etc. %;                                               6a2i1c1e
         cmlresults (N, result1, ... resultN); % re-
         turn N results - need not be called if not
         returning any results.  Results returned will
         be freed by the FE automatically%                     6a2i1c1f
         RETURN (dosuc, saveword);                             6a2i1c1g
         END;                                                  6a2i1c1h
      = terminate: % command is done, cleanup %                 6a2i1c2
      = abortcmd: % command was aborted, cleanup and
      restore state %                                           6a2i1c3
      = backup: % command was backed up %                       6a2i1c4
      ENDCASE ABORT (???, $"Bug:  Illegal reason to a
      FEFunction");                                             6a2i1c5
   RETURN (notme);                                               6a2i1d
   END.                                                          6a2i1e
      %%                                                        6a2i1e1

(for)                                                             6a2j
   FOR for-clause DO                                            6a2j1
      BEGIN                                                     6a2j1a
      END;                                                      6a2j1b

(grammar)                                                         6a2k
   %GR%   FILE SubSysName  % GRammar %                          6a2k1
      % COMPILE-INSTRUCTIONS %                                  6a2k1a
         INCLUDE <SsSrc, NLS-Grammar, flags !subsystems>
                                                               6a2k1a1
      % DECLARATIONS %                                         6a2k1b
```

```
    INCLUDE <SsSrc, NLS-Grammar, declarations
    !universal>                                          6a2k1b1
    DECLARE COMMAND WORD                                 6a2k1b2
       %subsystem command words (should be 100 to
       127)%                                             6a2k1b2a
       "COMMAND1" = 100,                                 6a2k1b2b
       "COMMANDn" = 10n-1;                               6a2k1b2c
       %%                                                6a2k1b2d
    DECLARE FUNCTION PROCESS = "PROCESSX" , PACKAGE
    = "PCKX" :                                           6a2k1b3
       xroutine1,                                        6a2k1b3a
       xroutinen;                                        6a2k1b3b
    DECLARE FEFUNCTION                                   6a2k1b4
       fefunc1,                                          6a2k1b4a
       fefuncn;                                          6a2k1b4b
    DECLARE PARSEFUNCTION                                6a2k1b5
       pffunc1,                                          6a2k1b5a
       pffuncn;                                          6a2k1b5b
    DECLARE GLOBAL                                       6a2k1b6
       global1,                                          6a2k1b6a
       globaln;                                          6a2k1b6b
    DECLARE VARIABLE                                     6a2k1b7
       variable1,                                        6a2k1b7a
       variablen;                                        6a2k1b7b
       %%                                                6a2k1b7c
% COMMON-RULES %                                         6a2k1c
    INCLUDE <SsSrc, NLS-Grammar, rules !universal>       6a2k1c1
                                                         6a2k1c1a
    rule1 = <rule-body>;                                 6a2k1c2
                                                         6a2k1c2a
    rulen = <rule-body>;                                 6a2k1c3
                                                         6a2k1c3a
       %%                                                6a2k1c3b
% COMMANDS % SUBSYSTEM SubSysName KEYWORD
"SUBSYSNAME"                                             6a2k1d
    INITIALIZATION % for SubSysName %                    6a2k1d1
       initsubsys = xinitsubsys();
                                                         6a2k1d1a
    TERMINATION % for SubSysName %                       6a2k1d2
       trmsubsys = xtrmsubsys();
                                                         6a2k1d2a
    command1 COMMAND = "COMMANDWORD1"                    6a2k1d3
       <rule-body>                                       6a2k1d3a
       xroutine1(variable1, variable2);
                                                         6a2k1d3b
    command2 COMMAND = "COMMANDWORD2"                    6a2k1d4
       <rule-body>                                       6a2k1d4a
       xroutine2(variable1, variable2);
                                                         6a2k1d4b
```

```
      INCLUDE <SsSrc, NLS-Grammar, commands
      !universal>                                          6a2k1d5
      END.                                                 6a2k1d6
   FINISH                                                  6a2k1e

(if-then-else)                                              6a21
   IF if-clause THEN                                        6a211
      BEGIN                                                 6a211a
      END                                                   6a211b
   ELSE                                                     6a212
      BEGIN                                                 6a212a
      END;                                                  6a212b

(jovial-program)                                            6a2m
   !COMPOOL! ('compool-file') name, name;                   6a2m1
   PROGRAM programname  " Description "                     6a2m2
      BEGIN                                                 6a2m2a
      " Program description "                               6a2m2b
      " DECLARATIONS "                                      6a2m2c
         " FUNCTIONS DEFINED "                              6a2m2c1
            " one-line-description "
            DEF PROC procname (input-parameter1, ...,
            input-parametern  : output parameter1, ...,
            output-parametern) functiontype ;               6a2m2c1a
               " Procedure description                      6a2m2c1a1
                  FUNCTION                                  6a2m2c1a1a
                     none                                   6a2m2c1a1a1
                  ARGUMENTS                                 6a2m2c1a1b
                     none                                   6a2m2c1a1b1
                  RESULTS                                   6a2m2c1a1c
                     none                                   6a2m2c1a1c1
                  NON-STANDARD CONTROL                      6a2m2c1a1d
                     none                                   6a2m2c1a1d1
                  "                                         6a2m2c1a1e
               BEGIN                                        6a2m2c1a2
               " Declarations "                             6a2m2c1a3
               "function body"                              6a2m2c1a4
               " Return "                                   6a2m2c1a5
                  RETURN;                                   6a2m2c1a5a
               END                                          6a2m2c1a6
         " SUBROUTINES DEFINED "                            6a2m2c2
            " one-line-description "
            REF PROC (procname) (input-parameter1, ...,
            input-parametern  : output parameter1, ...,
            output-parametern) ;                            6a2m2c2a
               " Procedure description                      6a2m2c2a1
                  FUNCTION                                  6a2m2c2a1a
                     none                                   6a2m2c2a1a1
                  ARGUMENTS                                 6a2m2c2a1b
```

```
                    none                                    6a2m2c2a1b1
                 RESULTS                                     6a2m2c2a1c
                    none                                    6a2m2c2a1c1
                 NON-STANDARD CONTROL                        6a2m2c2a1d
                    none                                    6a2m2c2a1d1
                 "                                           6a2m2c2a1e
              BEGIN                                           6a2m2c2a2
              " Declarations "                                6a2m2c2a3
              "procedure body"                                6a2m2c2a4
              " Return "                                      6a2m2c2a5
                 RETURN;                                     6a2m2c2a5a
              END                                             6a2m2c2a6
        " ITEMS DEFINED "                                       6a2m2c3
           ITEM itemname itemtype;                             6a2m2c3a
     "program body"                                             6a2m2d
     END                                                        6a2m2e
        %%                                                     6a2m2e1
(110-program)                                                     6a2n
   FILE ProgramName  % Description %                             6a2n1
     % DECLARATIONS %                                          6a2n1a
     %%                                                        6a2n1b
     % PROCEDURE %                                            6a2n1c
        (procname)   % CL: ;  one-line-description %
        PROCEDURE (arg1 <type>, ..., argn % => [meta-
        res] res1 <type>, ..., resn %);                       6a2n1c1
           % Procedure description                           6a2n1c1a
              FUNCTION                                      6a2n1c1a1
                 none                                      6a2n1c1a1a
              ARGUMENTS                                      6a2n1c1a2
                 none                                      6a2n1c1a2a
              RESULTS                                        6a2n1c1a3
                 none                                      6a2n1c1a3a
              NON-STANDARD CONTROL                           6a2n1c1a4
                 none                                      6a2n1c1a4a
              %                                             6a2n1c1a5
           % Declarations %                                  6a2n1c1b
           %procedure body%                                  6a2n1c1c
           % Return %                                        6a2n1c1d
              RETURN;                                       6a2n1c1d1
           END.                                             6a2n1c1e
              %%                                           6a2n1c1e1
     FINISH                                                   6a2n1d
(11011-program)                                                  6a2c
   FILE ProgramName  % Description %                           6a2o1
     % DECLARATIONS %                                         6a2o1a
     %%                                                       6a2o1b
     % PROCEDURE %                                           6a2o1c
```

```
        (procname)   % CL: ;  one-line-description %
        PROCEDURE (arg1 <type>, ..., argn % => [meta-
        res] res1 <type>, ..., resn %);                    6a2o1c1
           % Procedure description                         6a2o1c1a
              FUNCTION                                     6a2o1c1a1
                 none                                      6a2o1c1a1a
              ARGUMENTS                                    6a2o1c1a2
                 none                                      6a2o1c1a2a
              RESULTS                                      6a2o1c1a3
                 none                                      6a2o1c1a3a
              NON-STANDARD CONTROL                         6a2o1c1a4
                 none                                      6a2o1c1a4a
              %                                            6a2o1c1a5
           % Declarations %                                6a2o1c1b
           %procedure body%                                6a2o1c1c
           % Return %                                      6a2o1c1d
              RETURN;                                      6a2o1c1d1
           END.                                            6a2o1c1e
              %%                                           6a2o1c1e1
     FINISH                                                  6a2o1d
(loop)                                                       6a2p
   LOOP                                                      6a2p1
      BEGIN                                                  6a2p1a
      END;                                                   6a2p1b
(meta-program)                                               6a2q
   FILE filename CHECK                                       6a2q1
   META program                                              6a2q2
   % Compiler header.  %                                     6a2q3
    ERROR: ;                                                 6a2q4
    SIZE: ;                                                  6a2q5
    FLAGS: ;                                                 6a2q6
    DUMMY: ;                                                 6a2q7
    SET: ;                                                   6a2q8
    FIELDS: ;                                                6a2q9
    ATTRIBUTES: ;                                            6a2q10
    OPCODES: ;                                               6a2q11
   % Compiler header syntax. %                               6a2q12
   %  Rules. %                                               6a2q13
   END of TREE META                                          6a2q14
(code)                                                       6a2r
   %PFC% COROUTINE (parm1, parm2, parm3);                    6a2r1
      END.                                                   6a2r1a
(data)                                                       6a2s
   %PFD%  FILE SubSysName  % Parse/Fe functions Data %       6a2s1
      FINISH                                                 6a2s1a
(function)                                                   6a2t
   (ParseFunctionName)   % CL: ;  one-line-description %
   COROUTINE (reason, instruction, accumulator REF,
   argcount, arguments REF, saveword % => result %);       6a2t1
```

```
        % Parsefunction description                          6a2t1a
           FUNCTION                                          6a2t1a1
              none                                           6a2t1a1a
           ARGUMENTS (show grammar arguments)                6a2t1a2
              none                                           6a2t1a2a
           RESULT (show grammar results)                     6a2t1a3
              none                                           6a2t1a3a
           NON-STANDARD CONTROL                              6a2t1a4
              none                                           6a2t1a4a
           %                                                 6a2t1a5
        % Declarations %                                     6a2t1b
        % Initial entry point %                              6a2t1c
           PORT ENTRY                                        6a2t1c1
              % initialization %                             6a2t1c1a
           EXIT PCALL;                                       6a2t1c2
        %coroutine body%                                     6a2t1d
        END.                                                 6a2t1e
           %%                                                6a2t1e1
(parsefunction)                                               6a2u
   (ParseFunctionName)   % CL: ;  one-line-description %
   COROUTINE (reason, instruction, accumulator REF,
   argcount, arguments REF, saveword % => result %);         6a2u1
      % Parsefunction description                           6a2u1a
         FUNCTION                                          6a2u1a1
            none                                           6a2u1a1a
         ARGUMENTS (show grammar arguments)                6a2u1a2
            none                                           6a2u1a2a
         RESULT (show grammar results)                     6a2u1a3
            none                                           6a2u1a3a
         NON-STANDARD CONTROL                              6a2u1a4
            none                                           6a2u1a4a
         %                                                 6a2u1a5
      % Declarations %                                     6a2u1b
      % Initial entry point %                              6a2u1c
         PORT ENTRY                                        6a2u1c1
            % Initialization %                             6a2u1c1a
         EXIT PCALL;                                       6a2u1c2
      %coroutine body%                                     6a2u1d
      END.                                                 6a2u1e
         %%                                                6a2u1e1
(procedure)                                                   6a2v
   (procname)   % CL: ;  one-line-description %
   PROCEDURE (arg1 <type>, ..., argn % => [meta-res] res1
   <type>, ..., resn %);                                     6a2v1
      % Procedure description                               6a2v1a
         FUNCTION                                          6a2v1a1
            none                                           6a2v1a1a
         ARGUMENTS                                         6a2v1a2
            none                                           6a2v1a2a
```

```
        RESULTS                                                  6a2v1a3
           none                                                 6a2v1a3a
        NON-STANDARD CONTROL                                     6a2v1a4
           none                                                 6a2v1a4a
        %                                                        6a2v1a5
     % Declarations %                                            6a2v1b
     %procedure body%                                            6a2v1c
     % Return %                                                  6a2v1d
        RETURN;                                                 6a2v1d1
     END.                                                        6a2v1e
        %%                                                      6a2v1e1

(program)                                                          6a2w
   %GR%   FILE SubSysName  % GRammar %                           6a2w1
     % COMPILE-INSTRUCTIONS %                                    6a2w1a
        INCLUDE <SsSrc, NLS-Grammar, flags !subsystems>
                                                                6a2w1a1
     % DECLARATIONS %                                           6a2w1b
        INCLUDE <SsSrc, NLS-Grammar, declarations
        !universal>                                             6a2w1b1
        DECLARE COMMAND WORD                                    6a2w1b2
           %subsystem command words (should be 100 to
           127)%                                               6a2w1b2a
           "COMMAND1" = 100,                                   6a2w1b2b
           "COMMANDn" = 10n-1;                                 6a2w1b2c
           %%                                                  6a2w1b2d
        DECLARE FEFUNCTION                                      6a2w1b3
           fefunc1,                                            6a2w1b3a
           fefuncn;                                            6a2w1b3b
        DECLARE FUNCTION                                        6a2w1b4
           PROCESS = "PROCESSX" , PACKAGE = "PCKX" :          6a2w1b4a
              xroutine1,                                     6a2w1b4a1
              xroutinen;                                     6a2w1b4a2
        DECLARE GLOBAL                                          6a2w1b5
           global1,   % short description of global1 %
                                                              6a2w1b5a
           globaln;   % short description of globaln %
                                                              6a2w1b5b
        DECLARE VARIABLE                                        6a2w1b6
           variable1,                                         6a2w1b6a
           variablen;                                         6a2w1b6b
           %%                                                 6a2w1b6c
     % COMMON-RULES %                                          6a2w1c
        INCLUDE <SsSrc, NLS-Grammar, rules !universal>      6a2w1c1
                                                            6a2w1c1a
        rule1 = <rule-body>;                                 6a2w1c2
                                                            6a2w1c2a
        rulen = <rule-body>;                                 6a2w1c3
                                                            6a2w1c3a
```

```
        %%                                                    6a2w1c3b
    % COMMANDS % SUBSYSTEM SubSysName KEYWORD
    "SUBSYSNAME"                                              6a2w1d
        INITIALIZATION % for SubSysName %                     6a2w1d1
            initsubsys = xinitsubsys();
                                                              6a2w1d1a
        TERMINATION % for SubSysName %                        6a2w1d2
            trmsubsys = xtrmsubsys();
                                                              6a2w1d2a
        command1 COMMAND = "COMMANDWORD1"                     6a2w1d3
            <rule-body>                                       6a2w1d3a
            xroutine1(variable1, variable2);
                                                              6a2w1d3b
        command2 COMMAND = "COMMANDWORD2"                     6a2w1d4
            <rule-body>                                       6a2w1d4a
            xroutine2(variable1, variable2);
                                                              6a2w1d4b
        INCLUDE <SsSrc, NLS-Grammar, commands
        !universal>                                           6a2w1d5
        END.                                                  6a2w1d6
    FINISH                                                    6a2w1e

%PFC%  FILE SubSysName  % Parse/Fe functions Code %           6a2w2
    % DECLARATIONS %                                         6a2w2a
    % CODE %                                                 6a2w2b
        (FEFunctionName)  % CL: ;  one-line description
        %
        PROCEDURE (reason, instruction, accumulator REF,
        argcount, arguments REF, saveword % => result
        %);                                                   6a2w2b1
            % FEFunction description                        6a2w2b1a
                FUNCTION                                     6a2w2b1a1
                    none                                     6a2w2b1a1a
                ARGUMENTS                                    6a2w2b1a2
                    reason - reason fefunction is being
                    invoked                                  6a2w2b1a2a
                    instruction - byte pointer to grammar
                    instruction                              6a2w2b1a2b
                    accumulator - pointer to global
                    accumulator                              6a2w2b1a2c
                    argcount - count of number of arguments
                    from grammar call                        6a2w2b1a2d
                    arguments - pointer to array of argu-
                    ment values from grammar call            6a2w2b1a2e
                    saveword - word of context retained by
                    Frontend                                 6a2w2b1a2f
                RESULT                                       6a2w2b1a3
                    reason describing fefunction result
                                                             6a2w2b1a3a
```

```
      NON-STANDARD CONTROL                                  6a2w2b1a4
         none                                              6a2w2b1a4a
         %                                                  6a2w2b1a5
      % Declarations %                                       6a2w2b1b
      CASE reason OF                                         6a2w2b1c
         = parsing: % being invoked for first time
         during command %                                    6a2w2b1c1
            BEGIN                                           6a2w2b1c1a
            % decide whether FF is on the correct
            path through the grammar %                      6a2w2b1c1b
            IF % not on right path % THEN RETURN
            (notme);                                        6a2w2b1c1c
            % do processing %                               6a2w2b1c1d
            saveword _ % word of context to be
            saved or 0 - if 0, will not be called
            during backup, etc. %;                          6a2w2b1c1e
            cmlresults (N, result1, ... resultN); %
            return N results - need not be called
            if not returning any results.  Results
            returned will be freed by the FE auto-
            matically%                                      6a2w2b1c1f
            RETURN (dosuc, saveword);                       6a2w2b1c1g
            END;                                            6a2w2b1c1h
         = terminate: % command is done, cleanup %
                                                            6a2w2b1c2
         = abortcmd: % command was aborted, cleanup
         and restore state %                                6a2w2b1c3
         = backup: % command was backed up %                6a2w2b1c4
         ENDCASE ABORT (???, $"Bug:  Illegal reason
         to a FEFunction");                                 6a2w2b1c5
      RETURN;                                               6a2w2b1d
      END.                                                  6a2w2b1e
         %%                                                 6a2w2b1e1
(ParseFunctionName)   % CL: ;  one-line-
description %
COROUTINE (reason, instruction, accumulator REF,
argcount, arguments REF, saveword % => result
%);                                                         6a2w2b2
      % Parsefunction description                          6a2w2b2a
         FUNCTION                                          6a2w2b2a1
            none                                          6a2w2b2a1a
         ARGUMENTS                                         6a2w2b2a2
            reason - reason parsefunction is being
            invoked                                       6a2w2b2a2a
            instruction - byte pointer to grammar
            instruction                                   6a2w2b2a2b
            accumulator - pointer to global
            accumulator                                   6a2w2b2a2c
```

```
                argcount - count of number of arguments
                from grammar call                          6a2w2b2a2d
                arguments - pointer to array of argu-
                ment values from grammar call              6a2w2b2a2e
                saveword - word of context retained by
                Frontend                                   6a2w2b2a2f
             RESULT                                         6a2w2b2a3
                reason describing parsefunction result
                                                           6a2w2b2a3a
             NON-STANDARD CONTROL                           6a2w2b2a4
                none                                       6a2w2b2a4a
             %                                              6a2w2b2a5
          % Declarations %                                   6a2w2b2b
          % Initial entry point %                           6a2w2b2c
             PORT ENTRY                                     6a2w2b2c1
                % Initialization %                          6a2w2b2c1a
             EXIT PCALL;                                    6a2w2b2c2
          <coroutine body>                                  6a2w2b2d
          END.                                              6a2w2b2e
             %%                                             6a2w2b2e1
   FINISH                                                     6a2w2c

%PFD%  FILE SubSysName  % Parse/Fe functions Data %          6a2w3
   FINISH                                                   6a2w3a

%BE%   FILE SubSysName  % BackEnd %                         6a2w4
   % DECLARATIONS %                                         6a2w4a
      % Dispatch-Table: address of routine-name-
      string, address of routine %                          6a2w4a1
         (subsysname) EXTERNAL = (                          6a2w4a1a
            $"XROUTINE1", $xroutine1,                       6a2w4a1a1
            $"XROUTINEn", $xroutinen,                       6a2w4a1a2
            0,0);  % table must end with zero %            6a2w4a1a3
            %%                                              6a2w4a1a4
      % Command Words %                                     6a2w4a2
         (cw1) CONSTANT = 100B;                             6a2w4a2a
         (cwn) CONSTANT = 10n-1B;                           6a2w4a2b
         %%                                                 6a2w4a2c
   % X-ROUTINES %                                          6a2w4b
      (xroutine1)   % one-line description %                6a2w4b1
      (xroutinen)   % one-line description %                6a2w4b2
   % CORE-ROUTINES %                                       6a2w4c
      (croutine1)   % one-line description %                6a2w4c1
      (croutinen)   % one-line description %                6a2w4c2
   % SUPPORT-ROUTINES %                                    6a2w4d
      (sroutine1)   % one-line description %                6a2w4d1
      (sroutine1)   % one-line description %                6a2w4d2
   FINISH                                                   6a2w4e
```

```
(run-program)                                                    6a2x
   FILE ProgramName  % Description %                             6a2x1
      % DECLARATIONS %                                          6a2x1a

   (ProgramName)   % CL: ;  one-line-description %
   PROCEDURE (arg1 <type>, ..., argn % => [meta-res] res1
   <type>, ..., resn %);                                         6a2x2
      % Procedure description                                   6a2x2a
         FUNCTION                                              6a2x2a1
            none                                              6a2x2a1a
         ARGUMENTS                                             6a2x2a2
            none                                              6a2x2a2a
         RESULTS                                               6a2x2a3
            none                                              6a2x2a3a
         NON-STANDARD CONTROL                                  6a2x2a4
            none                                              6a2x2a4a
         %                                                     6a2x2a5
      % Declarations %                                         6a2x2b
      %procedure body%                                         6a2x2c
      % Return %                                              6a2x2d
         RETURN;                                              6a2x2d1
      END.                                                    6a2x2e
      FINISH %%                                               6a2x2f

(sequence-generator)                                             6a2y
   (nameofseqgen)   % CL: ;  one-line-description %
   PROCEDURE (sw REF, entrytype);                               6a2y1
      % Procedure description                                  6a2y1a
         FUNCTION                                             6a2y1a1
            This is a user sequence generator.  That . .
            .                                               6a2y1a1a
         ARGUMENTS                                            6a2y1a2
            sw - REF-address of sequence work area.  See
            record def, (nine, brecords, seqr).             6a2y1a2a
            entrytype - INTEGER-entry type.                 6a2y1a2b
               =sqopn:  called at seq open to initialize
               a work area                                 6a2y1a2b1
               =sqgnxt:  called for next in seq            6a2y1a2b2
               =sqcls:  called at seq close to release
               the workarea                                6a2y1a2b3
         RESULTS                                              6a2y1a3
            none                                            6a2y1a3a
         NON-STANDARD CONTROL                                 6a2y1a4
            When called with entry type sqgnxt, a pseudo
            coroutine is used for the sequence generator
            return mechanism.  Control is given up by
            calling send or sport, which eventually
            switches the stack.  Control is returned when
```

```
        a call is made to seqgen, which makes a
        coroutine port call to the stack associated
        with this sequence work area.                    6a2y1a4a
        A signal is generated (err is called) when
        the seq generator is called with an illegal
        entry type.                                      6a2y1a4b
        Note: for an example look at system sequence
        generator.                                       6a2y1a4c
     %                                                    6a2y1a5
  % Declarations %                                        6a2y1b
     % Caution!! Locals are not consistent across en-
     tries %                                             6a2y1b1
  % select entry point %                                  6a2y1c
     CASE entrytype OF                                   6a2y1c1
        =sqopn:   % called at seq open %                 6a2y1c1a
           NULL;                                         6a2y1c1a1
        =sqgnxt:  % call for next in seq %               6a2y1c1b
           LOOP                                          6a2y1c1b1
              BEGIN                                      6a2y1c1b1a
              % perform activities unique to this seq
              generator %                                6a2y1c1b1b
              % make "coroutine" call via call to
              send or sport %                            6a2y1c1b1c
              %   send($sw,     - addr of a sequence
              work area                                  6a2y1c1b1d
                     $str);     - an ENDFIL or addr
              of a string %                              6a2y1c1b1e
              %   sport($sw);  - addr of sequence
              work area %                                6a2y1c1b1f
              % returned here by sport call from
              seqgen %                                   6a2y1c1b1g
              % get next in sequence %                   6a2y1c1b1h
              END;                                       6a2y1c1b1i
        =sqcls:   % called at seq close %                6a2y1c1c
           NULL;                                         6a2y1c1c1
        ENDCASE err($"Bug");                             6a2y1c1d
  % Return %                                             6a2y1d
     RETURN;                                             6a2y1d1
  END.                                                   6a2y1e
     %%                                                  6a2y1e1

(sort-key)                                               6a2z
   (SortName) FILE                                       6a2z1
      ALLOW!                                             6a2z1a

   %Declarations%                                        6a2z2

   %...Default key procedure...%                         6a2z3
```

```
(defkey)   % CL: ;  one-line-description %
PROCEDURE (stid, % handle of the statement begin
considered %                                          6a2z3a
   buffer, % address of a buffer to hold the
   sorting index %                                    6a2z3a1
   bufflength); % the maximum size (in words) of
   the sorting index %                                6a2z3a2
   % Procedure description                            6a2z3a3
      FUNCTION                                       6a2z3a3a
         none                                       6a2z3a3a1
      ARGUMENTS                                      6a2z3a3b
         none                                       6a2z3a3b1
      RESULTS                                        6a2z3a3c
         none                                       6a2z3a3c1
      NON-STANDARD CONTROL                           6a2z3a3d
         none                                       6a2z3a3d1
      %                                              6a2z3a3e
   % Declarations %                                  6a2z3a4
   %procedure body%                                  6a2z3a5
   % Return %                                        6a2z3a6
      RETURN (partial-index-flag, % TRUE - data in
      buffer only high-order bits of the sorting
      index FALSE - buffer  value is full sorting
      index %                                       6a2z3a6a
      word-count); % integer indicating the number
      of words actually used in the buffer %        6a2z3a6b
   END.                                              6a2z3a7
      %%                                            6a2z3a7a
buffer, bufflength);                                  6a2z3b
                                                      6a2z3c

FINISH                                                6a2z4

(subsystem)                                           6a2aa
   %GR%   FILE SubSysName  % GRammar %                6a2aa1
      % COMPILE-INSTRUCTIONS %                        6a2aa1a
         INCLUDE <SsSrc, NLS-Grammar, flags !subsystems>
                                                     6a2aa1a1
      % DECLARATIONS %                               6a2aa1b
         INCLUDE <SsSrc, NLS-Grammar, declarations
         !universal>                                 6a2aa1b1
         DECLARE COMMAND WORD                        6a2aa1b2
            %subsystem command words (should be 100 to
            127)%                                  6a2aa1b2a
            "COMMAND1" = 100,                      6a2aa1b2b
            "COMMANDn" = 10n-1;                    6a2aa1b2c
            %%                                     6a2aa1b2d
         DECLARE FEFUNCTION                          6a2aa1b3
            fefunc1,                               6a2aa1b3a
```

```
      fefuncn;                                                  6a2aa1b3b
   DECLARE FUNCTION                                              6a2aa1b4
      PROCESS = "PROCESSX" , PACKAGE = "PCKX" :                  6a2aa1b4a
         xroutine1,                                             6a2aa1b4a1
         xroutinen;                                             6a2aa1b4a2
   DECLARE GLOBAL                                                6a2aa1b5
      global1,   % short description of global1 %
                                                                 6a2aa1b5a
      globaln;   % short description of globaln %
                                                                 6a2aa1b5b
   DECLARE VARIABLE                                              6a2aa1b6
      variable1,                                                 6a2aa1b6a
      variablen;                                                 6a2aa1b6b
      %%                                                         6a2aa1b6c
% COMMON-RULES %                                                  6a2aa1c
   INCLUDE <SsSrc, NLS-Grammar, rules !universal>
                                                                 6a2aa1c1
                                                                 6a2aa1c1a
   rule1 = <rule-body>;                                          6a2aa1c2
                                                                 6a2aa1c2a
   rulen = <rule-body>;                                          6a2aa1c3
                                                                 6a2aa1c3a
      %%                                                         6a2aa1c3b
% COMMANDS % SUBSYSTEM SubSysName KEYWORD
"SUBSYSNAME"                                                      6a2aa1d
   INITIALIZATION % for SubSysName %                             6a2aa1d1
      initsubsys = xinitsubsys();
                                                                 6a2aa1d1a
   TERMINATION % for SubSysName %                                6a2aa1d2
      trmsubsys = xtrmsubsys();
                                                                 6a2aa1d2a
   command1 COMMAND = "COMMANDWORD1"                             6a2aa1d3
      <rule-body>                                                6a2aa1d3a
      xroutine1(variable1, variable2);
                                                                 6a2aa1d3b
   command2 COMMAND = "COMMANDWORD2"                             6a2aa1d4
      <rule-body>                                                6a2aa1d4a
      xroutine2(variable1, variable2);
                                                                 6a2aa1d4b
   INCLUDE <SsSrc, NLS-Grammar, commands
   !universal>                                                   6a2aa1d5
   END.                                                          6a2aa1d6
FINISH                                                           6a2aa1e

%PFC%  FILE SubSysName  % Parse/Fe functions Code %              6a2aa2
   % DECLARATIONS %                                             6a2aa2a
   % CODE %                                                     6a2aa2b
      (FEFunctionName)  % CL: ;  one-line description
      %
```

```
PROCEDURE (reason, instruction, accumulator REF,
argcount, arguments REF, saveword % => result
%);                                              6a2aa2b1
   % FEFunction description                      6a2aa2b1a
      FUNCTION                                   6a2aa2b1a1
         none                                    6a2aa2b1a1a
      ARGUMENTS                                  6a2aa2b1a2
         reason - reason fefunction is being
         invoked                                 6a2aa2b1a2a
         instruction - byte pointer to grammar
         instruction                             6a2aa2b1a2b
         accumulator - pointer to global
         accumulator                             6a2aa2b1a2c
         argcount - count of number of arguments
         from grammar call                       6a2aa2b1a2d
         arguments - pointer to array of argu-
         ment values from grammar call           6a2aa2b1a2e
         saveword - word of context retained by
         Frontend                                6a2aa2b1a2f
      RESULT                                     6a2aa2b1a3
         reason describing fefunction result
                                                 6a2aa2b1a3a
      NON-STANDARD CONTROL                       6a2aa2b1a4
         none                                    6a2aa2b1a4a
      %                                          6a2aa2b1a5
   % Declarations %                              6a2aa2b1b
   CASE reason OF                                6a2aa2b1c
      = parsing: % being invoked for first time
      during command %                           6a2aa2b1c1
         BEGIN                                   6a2aa2b1c1a
         % decide whether FF is on the correct
         path through the grammar %              6a2aa2b1c1b
         IF % not on right path % THEN RETURN
         (notme);                                6a2aa2b1c1c
         % do processing %                       6a2aa2b1c1d
         saveword _ % word of context to be
         saved or 0 - if 0, will not be called
         during backup, etc. %;                  6a2aa2b1c1e
         cmlresults (N, result1, ... resultN); %
         return N results - need not be called
         if not returning any results.  Results
         returned will be freed by the FE auto-
         matically%                              6a2aa2b1c1f
         RETURN (dosuc, saveword);               6a2aa2b1c1g
         END;                                    6a2aa2b1c1h
      = terminate: % command is done, cleanup %
                                                 6a2aa2b1c2
      = abortcmd: % command was aborted, cleanup
      and restore state %                        6a2aa2b1c3
```

```
            = backup: % command was backed up %   6a2aa2b1c4
            ENDCASE ABORT (???, $"Bug:  Illegal reason
            to a FEFunction");                    6a2aa2b1c5
         RETURN;                                   6a2aa2b1d
         END.                                      6a2aa2b1e
            %%                                     6a2aa2b1e1
      (ParseFunctionName)   % CL: ;  one-line-
      description %
      COROUTINE (reason, instruction, accumulator REF,
      argcount, arguments REF, saveword % => result
      %);                                           6a2aa2b2
         % Parsefunction description               6a2aa2b2a
            FUNCTION                               6a2aa2b2a1
               none                               6a2aa2b2a1a
            ARGUMENTS                              6a2aa2b2a2
               reason - reason parsefunction is being
               invoked                             6a2aa2b2a2a
               instruction - byte pointer to grammar
               instruction                         6a2aa2b2a2b
               accumulator - pointer to global
               accumulator                         6a2aa2b2a2c
               argcount - count of number of arguments
               from grammar call                   6a2aa2b2a2d
               arguments - pointer to array of argu-
               ment values from grammar call       6a2aa2b2a2e
               saveword - word of context retained by
               Frontend                            6a2aa2b2a2f
            RESULT                                 6a2aa2b2a3
               reason describing parsefunction result
                                                   6a2aa2b2a3a
            NON-STANDARD CONTROL                   6a2aa2b2a4
               none                               6a2aa2b2a4a
            %                                      6a2aa2b2a5
         % Declarations %                          6a2aa2b2b
         % Initial entry point %                   6a2aa2b2c
            PORT ENTRY                             6a2aa2b2c1
               % Initialization %                  6a2aa2b2c1a
            EXIT PCALL;                            6a2aa2b2c2
         <coroutine body>                          6a2aa2b2d
         END.                                      6a2aa2b2e
            %%                                     6a2aa2b2e1
   FINISH                                             6a2aa2c

%PFD%  FILE SubSysName  % Parse/Fe functions Data %   6a2aa3
   FINISH                                             6a2aa3a

%BE%   FILE SubSysName  % BackEnd %                   6a2aa4
   % DECLARATIONS %                                   6a2aa4a
```

```
      % Dispatch-Table: address of routine-name-
      string, address of routine %                    6a2aa4a1
         (subsysname) EXTERNAL = (                    6a2aa4a1a
            $"XROUTINE1", $xroutine1,                 6a2aa4a1a1
            $"XROUTINEn", $xroutinen,                 6a2aa4a1a2
            0,0);  % table must end with zero %        6a2aa4a1a3
            %%                                        6a2aa4a1a4
         % Command words %                             6a2aa4a2
            (cw1) CONSTANT = 100B;                    6a2aa4a2a
            (cwn) CONSTANT = 10n-1B;                  6a2aa4a2b
            %%                                        6a2aa4a2c
      % X-ROUTINES %                                   6a2aa4b
         (xroutine1)   % one-line description %       6a2aa4b1
         (xroutinen)   % one-line description %       6a2aa4b2
      % CORE-ROUTINES %                                6a2aa4c
         (croutine1)   % one-line description %       6a2aa4c1
         (croutinen)   % one-line description %       6a2aa4c2
      % SUPPORT-ROUTINES %                            6a2aa4d
         (sroutine1)   % one-line description %       6a2aa4d1
         (srcutine1)   % one-line description %       6a2aa4d2
      FINISH                                          6a2aa4e

(support-module)                                        6a2ab
   % SUP %   FILE ProgramName  % Support Routine %     6a2ab1
      % DECLARATIONS %                                6a2ab1a
      %%                                              6a2ab1b
      % PROCEDURE %                                   6a2ab1c
         (procname)   % CL: ;  one-line-description %
         PROCEDURE (arg1 <type>, ..., argn % => [meta-
         res] res1 <type>, ..., resn %);              6a2ab1c1
            % Procedure description                  6a2ab1c1a
               FUNCTION                              6a2ab1c1a1
                  none                               6a2ab1c1a1a
               ARGUMENTS                             6a2ab1c1a2
                  none                               6a2ab1c1a2a
               RESULTS                               6a2ab1c1a3
                  none                               6a2ab1c1a3a
               NON-STANDARD CONTROL                  6a2ab1c1a4
                  none                               6a2ab1c1a4a
               %                                     6a2ab1c1a5
            % Declarations %                         6a2ab1c1b
            %procedure body%                         6a2ab1c1c
            % Return %                               6a2ab1c1d
               RETURN;                               6a2ab1c1d1
            END.                                     6a2ab1c1e
               %%                                    6a2ab1c1e1
      % PROCEDURE %                                   6a2ab1d
```

```
            (procname)   % CL: ;  one-line-description %
            PROCEDURE (arg1 <type>, ..., argn % => [meta-
            res] res1 <type>, ..., resn %);                    6a2ab1d1
               % Procedure description                         6a2ab1d1a
                  FUNCTION                                     6a2ab1d1a1
                     none                                      6a2ab1d1a1a
                  ARGUMENTS                                    6a2ab1d1a2
                     none                                      6a2ab1d1a2a
                  RESULTS                                      6a2ab1d1a3
                     none                                      6a2ab1d1a3a
                  NON-STANDARD CONTROL                         6a2ab1d1a4
                     none                                      6a2ab1d1a4a
                  %                                            6a2ab1d1a5
               % Declarations %                                6a2ab1d1b
               %procedure body%                                6a2ab1d1c
               % Return %                                      6a2ab1d1d
                  RETURN;                                      6a2ab1d1d1
               END.                                            6a2ab1d1e
                  %%                                           6a2ab1d1e1
         FINISH                                                6a2ab1e

      (until-do)                                               6a2ac
         UNTIL until-clause DO                                 6a2ac1
            BEGIN                                              6a2ac1a
            END;                                               6a2ac1b

      (while-do)                                               6a2ad
         WHILE while-clause DO                                 6a2ad1
            BEGIN                                              6a2ad1a
            END;                                               6a2ad1b

   (jovial)                                                    6a3

      (directive)                                              6a3a
         !COMPOOL 'compool-file' name, name ;                  6a3a1

      (source)                                                 6a3b
         COMPOOL compool-name ;                                6a3b1
            BEGIN                                              6a3b1a
            END                                                6a3b1b

      (by-while)                                               6a3c
         FOR index: initial BY increment WHILE index relational
         stopvalue;                                            6a3c1
            BEGIN                                              6a3c1a
            END                                                6a3c1b

      (then-while)                                             6a3d
         FOR item-name THEN formula WHILE conditional-formula ; 6a3d1
```

```
      statement ;                                                6a3d1a

(if-else)                                                        6a3e
   IF conditional-formula ;                                      6a3e1
      statement ;                                                6a3e1a

   ELSE else-statement ;                                         6a3e2

(call)                                                           6a3f
   procedure-name@data-base(input-parameter, input-
   parameter : output-parameter, output-parameter) ;
   "comment"                                                     6a3f1

(declaration)                                                    6a3g
   PROC procedure-name data-allocator(input-parameter,
   input-parameter : output-parameter, output-parameter)
   ;                                                             6a3g1
      BEGIN                                                      6a3g1a
      "DECLARATIONS"                                             6a3g1b
      "PROCEDURE BODY"                                           6a3g1c
      END                                                        6a3g1d

(program)                                                        6a3h
   !COMPOOL! ('J73IO.CMP');                                      6a3h1

   PROGRAM programname  " Description "                          6a3h2
      BEGIN                                                      6a3h2a
      " Program description "                                    6a3h2b
      " DECLARATIONS "                                           6a3h2c
         " EXTERNAL PROCEDURES "                                 6a3h2c1
            " one-line-description "
            REF PROC procname (input-parameter1, ...,
            input-parametern  : output parameter1, ...,
            output-parametern) ;                                 6a3h2c1a
               " Procedure description                           6a3h2c1a1
                  FUNCTION                                       6a3h2c1a1a
                     none                                        6a3h2c1a1a1
                  ARGUMENTS                                      6a3h2c1a1b
                     none                                        6a3h2c1a1b1
                  RESULTS                                        6a3h2c1a1c
                     none                                        6a3h2c1a1c1
                  NON-STANDARD CONTROL                           6a3h2c1a1d
                     none                                        6a3h2c1a1d1
                  "                                              6a3h2c1a1e
               BEGIN                                             6a3h2c1a2
               " Declarations "                                  6a3h2c1a3
               END                                               6a3h2c1a4
         " SUBROUTINES DEFINED "                                 6a3h2c2
            " one-line-description "
```

```
                DEF PROC (procname) (input-parameter1, ...,
                input-parametern  : output parameter1, ...,
                output-parametern) ;                          6a3h2c2a
                   " Procedure description                    6a3h2c2a1
                      FUNCTION                                6a3h2c2a1a
                         none                                 6a3h2c2a1a1
                      ARGUMENTS                               6a3h2c2a1b
                         none                                 6a3h2c2a1b1
                      RESULTS                                 6a3h2c2a1c
                         none                                 6a3h2c2a1c1
                      NON-STANDARD CONTROL                    6a3h2c2a1d
                         none                                 6a3h2c2a1d1
                      "                                       6a3h2c2a1e
                   BEGIN                                      6a3h2c2a2
                   " Declarations "                           6a3h2c2a3
                   "procedure body"                           6a3h2c2a4
                   " Return "                                 6a3h2c2a5
                      RETURN;                                 6a3h2c2a5a
                   END                                        6a3h2c2a6
             " ITEMS DEFINED "                                  6a3h2c3
                ITEM itemname itemtype;                        6a3h2c3a
          "program body"                                         6a3h2d
          END                                                    6a3h2e
             %%                                                 6a3h2e1

   (declaration)                                                  6a3i
      PROGRAM program-name ;                                     6a3i1
         BEGIN                                                  6a3i1a
         "DECLARATIONS"                                         6a3i1b
         "PROGRAM BODY"                                         6a3i1c
         END "program-name"                                     6a3i1d

   (switch)                                                       6a3j
      SWITCH numeric-formula ;                                   6a3j1
         BEGIN                                                  6a3j1a
         [] statement ;                                         6a3j1b
         [] statement ;                                         6a3j1c
         [] statement ;                                         6a3j1d
         END                                                    6a3j1e

   (while)                                                        6a3k
      WHILE conditional-formula ;                                6a3k1
         controlled-statement ;                                 6a3k1a

(meta)                                                            6a4

   (program)                                                      6a4a
      FILE filename  % one line comment %                        6a4a1
         META program                                           6a4a1a
```

Appendix 2: PROGRAMS Subsystem Templates

```
    % COMPILER HEADER %                                  6a4a1a1
       DUMMY:                                           6a4a1a1a
       ERROR:                                           6a4a1a1b
       FLAGS:                                           6a4a1a1c
       ATTRUBUTES:                                      6a4a1a1d
       OPCODES:                                         6a4a1a1e
       SET:                                             6a4a1a1f
       SIZE:                                            6a4a1a1g
       FIELDS:                                          6a4a1a1h
    % SYNTAX RULES %                                     6a4a1a2
       identifier = rulebody;                           6a4a1a2a
    % PRODUCTION RULES %                                 6a4a1a3
       identifier[test-expression] =>                   6a4a1a3a
          production-rule-body;                        6a4a1a3a1
       identifier[test-expression] =>                   6a4a1a3b
          production-rule-body;                        6a4a1a3b1
    % VALUE RULES %                                      6a4a1a4
       identifier[test-expression] :=                   6a4a1a4a
          value-rule-body;                             6a4a1a4a1
       identifier[test-expression] :=                   6a4a1a4b
          value-rule-body;                             6a4a1a4b1
FINISH                                                   6a4a1b
```

Appendix 3: Command Sequence Processor Design Specifications:
AUGMENT Journal (29046,) Donald I. Andrews, 28 January 1979 7

COMMAND SEQUENCE PROCESSOR DESIGN SPECIFICATIONS 7a

Preface 7b

This report is a user-level description of an National Software Works facility for writing and executing Command Sequences for NSW tools and the NSW EXEC. It was prepared for the RADC NSW project, Contract F 30602-75-C-0320 7b1

Introduction 7c

A Command Sequence is a collection of one or more commands with a unique name. The user invokes a Command Sequence by its name; the NSW Frontend then processes the Sequence as if the user were typing in that collection of commands himself. The commands available for Command Sequence use include the NSW EXEC commands, all split tool commands, and unsplit tool input--in short, everything the user is allowed to do in the NSW, including the use of other Command Sequences. This Frontend feature and its associated program modules are called the Command Sequence Processor. 7c1

The great advantage of a Command Sequence facility, of course, is that it allows users to "program" in the command language with which they are familiar; that is, they can specify a series of operations and have this "Program" executed at any time. No programming language must be learned. Although this kind of facility is available on many time-sharing systems, it is generally missing the control constructs (e.g., IF, FOR, CASE) so heavily used in algorithmic languages. The NSW Command Sequence Processor includes control features, and hence provides a complete language for "command programming". 7c2

The CLI grammar-driven interface system, with its recognition modes, feedback and noise words, and help features, is a significant improvement in making man-machine interfaces coherent and natural. The inclusion of the Command Sequence facility complements a powerful system by bringing programming-like capabilities into the user interface. 7c3

Capabilities 7d

The Command Sequence Processor (CSP) will have three basic capabilities: 7d1

Running "canned" command strings, in the same manner as TENEX Runfile. 7d1a

Collecting selections from the user at Command Sequence execution time. These may be used (perhaps more than once during an execution) as user supplied arguments in the canned commands. 7d1b

Testing conditions and doing different things based on the outcome. The conditions may be user input, variables, or the results of commands. Further, control constructs allow sequences to loop over a group of commands until a specified condition is met. 7d1c

The CSP operates independently of the Command Language Interpreter, and hence functions across the EXEC and all tools. The sequences may contain commands to run a tool, followed by commands for the tool, followed by more EXEC commands, and so forth. 7d2

What the User Needs to Know 7e

A minimum of information is required to use the CSP. The user need not learn a new language since a Command Sequence is constructed of user-level commands, written in textual form exactly as he would see them when executing such commands. The command words are written in full; noise words may or may not be present. 7e1

To have the Command Sequence performed, the user executes a CSP command (available at the EXEC and all split tools), specifying a Command Sequence name and arguments, if any. As an alternative, he can have the sequence name available as a top-level command in all grammars (EXEC and all split tools). 7e2

Although Command Sequences cannot be invoked from an unsplit tool without escaping back to the EXEC, they can specify commands for unsplit tools. In that case, the Command Sequence will contain the text that would be typed to the tool from the terminal, which may not be as readable for the user

as Command Sequences for split tools. This is unavoidable since the command recognition mode of unsplit tools in general cannot be controlled. 7e3

To take advantage of advanced CSP capabilities, the user must know the syntax and semantics of the control constructs necessary to obtain selections from him, test variables, perform looping, etc. These constructs are as simple and intuitive as possible. 7e4

Command Sequence Generation 7f

A Command Sequence may be generated in several ways. The most obvious is to write the text or retrieve and edit an existing Command Sequence with an editor. However, there are more convienent methods. 7f1

A Command Sequence can be generated with the aid of the CSP itself. Basically, the user executes a CSP command to begin recording a Command Sequence. From that point on, every bit of input the user gives is incorporated into the Command Sequence. This includes EXEC, split, and unsplit tool commands. This continues until the user terminates the recording with another CSP command. The result is a Command Sequence that can be invoked immediately or stored for future use. 7f2

At "start recording" time the user may specify whether or not to actually execute the following (recorded) commands; that is, he may generate a Command Sequence without actually executing any commands. If unsplit tool commands are given in this mode, there is no feedback from the tool, since it is not really executing. For split tool and EXEC commands, the command feedback is exactly as if the command were executed, except for messages that would come from the tool itself or from the Works Manager. 7f3

The Command Sequence Generator Tool may also be used to originate a Command Sequence. This tool aids the user while he steps through the commands for his sequence. It simulates the "recording without executing" case above, but makes it possible to specify user input collection, testing, branching, and looping points within the sequence. The result is again a Command Sequence ready for use with the CSP. This tool also has a "debugging" mode whereby Command Sequences can be executed in slow motion and modified if necessary. 7f4

Command Sequence Control Constructs 7g

This section is intended to give the reader an idea of the form and capabilities of the Command Sequence control constructs. It may be incomplete in some respects. The control escape character is printed as an exclaimation mark (!). Command words are capitalized and user input is indicated inside angle brackets (< and >). Noise words are in parentheses. 7g1

All of the following commands may be executed when the user is in any split tool or at the NSW EXEC. They may also appear in Command Sequences. Note that the "canned" Command Sequence capability can be used when only the "recording" and "executing" commands are known. The other control commands are for more advanced capabilities. 7g2

Executing Command Sequences 7g3

!Do (CS name) <name> 7g3a

This command causes the Frontend to execute commands in the Command Sequence named "name". 7g3a1

Recording Command Sequences 7g4

!Start Recording (CS name) <name> 7g4a

<commands to be in Command Sequence "name"> 7g4b

!Stop Recording 7g4c

The Start and Stop commands are used to create a Command Sequence named "name". 7g4c1

Getting Selections from the User and Showing Strings 7g5

!Text (from user into) <varname> 7g5a

!Character (from user into) <varname> 7g5b

!word (from user into) <varname> 7g5c

The above commands cause the Command Sequence Processor to collect the specified kind of selection from the user rather than obtaining it from the Command Sequence text. After collecting the selection from the user, the CSP stores the input in the named variable

("varname" here) and directs its attention back to the Command Sequence text. These commands might be used to allow the user to specify a file name or other argument in a command the Command Sequence is performing. 7g5c1

!Selection (from user into) <selvar> 7g5d

This command collects a specific kind of selection from the user. It is used when the Command Sequence writer wants to get user input appropriate for a command in his Sequence. To do so, he specifies that command, using the "Selection" command at the point in which user input is required. The Selection command then looks at the grammar for the command he is specifying to determine what kind of selection is needed from the user. The selection input is stored in "selvar" and at the same time is provided as input to the selection instruction. In the following example the user will give a selection for the Insert Statement command. The selection will be saved in variable "place" for possible later use: 7g5d1

```
Insert Statement !Selection (from user into)
<place> <CA>                                    7g5d1a
```

!(<noise words>) 7g5e

The "noise words" command () puts the given text in the command feedback line, allowing the CS writer to prompt the user with text strings. (Syntax note: In this command the parentheses indicate what the user inserts in his Command Sequence rather than noise words displayed by a command.) The following illustrates the use of this command. It shows how to collect a string from the user and save it: 7g5f

```
!( <some text> )                                7g5f1

!Text (from user into) <save>                   7g5f2
```

Conditionals, Looping 7g6

The following commands control the CSP's path over the Command Sequence. Their arguments are single commands. 7g6a

!Begin (command group) <commands> !End 7g6b

The Begin and End commands make several commands (those grouped between the commands) into a single command. 7g6b1

!If <command> (then) <command> (else) <command> 7g6c

The If command permits the execution of either one command or another, based on the result of the first command. Every command has a True or False result which determines whether the (then/True) or (else/False) commands are executed. A command always has the result TRUE unless on of the following happens: 7g6c1

A remote (backend) call returns a failure result. 7g6c1a

A global variable is explicitly set by the grammar during command execution (the variable's name is not specified at this time). 7g6c1b

The CLI aborts the command for some reason. 7g6c1c

The command is "!FALSE". 7g6c1d

!Loop <command> 7g6d

The Loop command causes the specified "command" to be executed repeatedly. 7g6d1

!Exit (loop) 7g6e

This command causes the CSP to stop performing the innermost loop and continue with the command following the Loop command. 7g6e1

!Repeat Loop 7g6f

This causes the CSP to start over at the first command of the current loop being performed. 7g6f1

Variables 7g7

!Define (variable named) <varname> 7g7a

!Local (variable named) <locname> 7g7b

The Define command defines a "global" variable; that is, the variable can be used by any Command Sequence. A "local" variable, defined by the Local command, is

used only in one specific Command Sequence and will be deleted when that Sequence is completed. A variable defined by either command may be an integer, string, or boolean. The type is determined when a value is assigned to it. 7g7b1

!Assign (variable) <varname> (_) <expression> 7g7c

The Assign command assigns a value and type to the variable (either global or local). The expression is made up of variables, user selections, and operators. (Although the operators are not specified at this time, they will include addition, subtraction, and string concatenation. 7g7c1

!Test (variable) <varname> 7g7d

!Relation <expression> <relation> <expression> 7g7e

The Test and Relation commands are intended to be used as commands within the if commands. The Test command simply sets a condition flag based on the variable. True/False results will be defined for all variable types. The Relation comand applies the relational operator to two expressions--the Relation command is TRUE if and only if the specified relation is TRUE. 7g7e1

!name 7g7f

This command identifies a global or local name. It may be a variable name or a Command Sequence name. The Command Sequence will be executed or, if the variable contains a string, treated as a Command Sequence and executed. If the variable is not a string, a condition flag will be set. 7g7f1

Miscellaneous 7g8

% <some text> % 7g8a

A comment may be inserted in the Command Sequence text anywhere a space is allowed, by surrounding the comment with percent signs. 7g8a1

!Null 7g8b

The Null command does nothing and always has the value TRUE. 7g8b1

!False 7g8c

The False command does nothing and always has the value FALSE. 7g8c1

!Echo (y/n) <Y or N> 7g8d

The Echo command determines whether or not the user sees the normal command feedback text at his terminal while the Command Sequence is being executed. Echo Y will result in showing the user the feedback for each command as it is executed. Echo N will result in showing the user only the Command Sequence noise words defined by the noise word command. 7g8d1

Appendix 4: Proposed External Design for a Process System (PS):
AUGMENT Journal (47188,), (Revised) Bruce L. Parsley 8

INTRODUCTION 8a

This document proposes an external design for a Process System (PS). This includes the user interface -- primarily what relevant AUGMENT commands there are -- and a new artificial language called the Process Language (PL). 8a1

The basic construct with which we are concerned here is called a "Process Command Sequence" or "PCS". This is basically a sequence of AUGMENT commands as described in the text of statements in an AUGMENT file. PCSs are basically the same as the old Process Commands, but with several additional features. 8a2

Almost nothing is said in this document about HOW this proposed design might be implemented, only external specifications are proposed here. Note also that initially it would not be necessary to implement this design in full: there are several features that could be added after the initial implementation. 8a3

Not much effort has been taken to make this document easy to understand because of its role as a draft language syntax. Further revisions are expected containing examples and explanations where appropriate. 8a4

USER INTERFACE 8b

There are two ways to invoke a Process Command Sequence: with the AUGMENT Process command and via a "User Command". 8b1

NB: The meta-language used in this section to describe the syntax of AUGMENT commands is something of a mix between CML and the meta-language used in ARC's user documentation, plus some informalities of my own devising. I hope it's understandable. 8b1a

Process command: 8b2

This command, which is a universal command rather than just a BASE command, has the following syntax: 8b2a

Process FSTRUCTURE (at) SSEL (CONFIRM / PARAMETERS CONFIRM)
8b2a1

FSTRUCTURE = the "filtered structure" that will replace STRUCTURE
8b2a1a

PARAMETERS = see next section 8b2a1b

Note that this is nearly the same as the old Process command with the added possibility of parameters, which are discussed in the next section. 8b2b

User Commands: 8b3

There is a facility for users to define and invoke their own User Commands. 8b3a

Defining User Commands: 8b3b

A new command is added to the BASE subsystem with the following syntax: 8b3b1

Define User-command (at) SSEL(Branch) (with level) (OK/1/2) CONFIRM 8b3b1a

NB: The "OK" above will act the same as a "1". 8b3b1a1

The indicated branch is assumed to be a properly formed PCS. In addition it must start with a "label" (see next section for the definition of a label). The label is examined and used as the associated command word. 8b3b2

Note that such User Commands are only "defined" for that session. In subsequent sessions with AUGMENT the User Commands will be unknown. 8b3b2a

There is also one or more new commands in the USEROPTIONS subsytem that will provide for the definition of User Commands. User Commands defined in this manner will be recognized in all subsequent AUGMENT sessions until the user deletes/excludes the definition, e.g., by using another command in USEROPTIONS. 8b3b3

This USEROPTIONS feature is analagous to USEROPTIONS' Include (subsystem/program) feature. 8b3b3a

Invoking User Commands: 8b3c

After a User Command has been "defined" by any of the BASE or USEROPTIONS commands, the FrontEnd CLI will act as if "label" were a universal command at the specified level. Then any time the user inputs the proper character(s) at the base command state of any subsystem, the CLI will act exactly as if the user had input the following: 8b3c1

Process Branch (at) SSEL(Branch) <OK> 8b3c1a

PROCESS LANGUAGE 8c

Introduction 8c1

Following is a complete, formal description of the syntax of the proposed Process Language (PL). 8c1a

The meta-languge in which the description is written is ARC's usual version of BNF with the following additions: 8c1b

"#<foo, bar>" is equivalent to "foo $(bar foo)" and
"$<foo, bar>" is equivalent to "[foo $(bar foo)]", 8c1b1

i.e., a sequence of n foos separated by n-1 bars; 8c1b1a

"SP" means the character with code 40B (space); 8c1b2

"CA" means the character with code 04B (control-D, Command-Accept); 8c1b3

"CD" means the character with code 30B (control-X, Command-Delete); 8c1b4

"CH" means any character except SP, CA, or '¦; 8c1b5

"CH -> some character" means any number of characters terminated by, but not including, the specified character. Thus, the modified BNF description of Label below, "Label = CH -> ':;" says that a label is any number (greater than or equal to one) of characters terminated by (but not including) a colon. 8c1b6

Note that PL is a fully typed and type-checked language. PL is interpreted rather than compiled. 8c1c

Some information about the semantics is included. Only things that are novel or that might be obscure are discussed. Readers are assumed to be familiar with typed, block-structured computer languages. 8c1d

```
ProcessCommandSequence = #Command;                                    8c2

   Command = NLSCommand / ProcessCommand ;                            8c2a

NLSCommand = #CommandWord
 $(Selection/Parameter/Confirm/YesNo/NoiseWords/LevAdj/ViewSpecs)
 ['¦ Result];                                                         8c3

   CommandWord = CH -> SP / '¦ [ CommandWordLHS '_ / CommandWordExp ] '¦ ;
                                                                      8c3a
   Selection   = CH -> CA / '¦ [ SelectionLHS   '_ / SelectionExp   ] '¦ ;
                                                                      8c3b
   Parameter   = CH -> CA / '¦ [ TextLHS        '_ / TextExp        ] '¦ ;
                                                                      8c3c
   Confirm     = CA / CD /  '¦ [ BooleanLHS     '_ / BooleanExp     ] '¦ ;
                                                                      8c3d
   YesNo       = CA / ('Y/'y) -> SP / ('N/'n) -> SP /
    '¦ [ BooleanLHS '_ / BooleanExp ] '¦ ;                            8c3e

   NoiseWords  = '( -> ')    / "¦(" [ TextLHS '_ / TextExp ] ")¦" ;
                                                                      8c3f
```

```
LevAdj      = $('d/'u) [CA] / '¦ [ TextLHS '_ / TextExp ] '¦ ;
                                                              8c3g
ViewSpecs   = CA / CH -> CA / '¦ [ TextLHS '_ / TextExp ] '¦ ;
                                                              8c3h
Result = ': BooleanLHS;                                       8c3i
```

In the syntax for the NLSCommands, the alternatives before the last are meant to represent the old Process Commands stuff. Note that it is not quite accurate or complete, e.g., a user may have changed his/her Command-Accept character, a CD occuring almost anywhere would screw up the "parse". 8c3j

The semantics of alternatives of the form
'¦ [LeftHandSide '_] [Expression] '¦
is as follows: 8c3k

¦¦ 8c3k1

Characters are taken from the user until the "thing" the CLI is looking for is complete. 8c3k1a

¦LeftHandSide_¦ 8c3k2

Characters are taken from the user until the "thing" the CLI is looking for is complete and the value of the user response is stored in the indicated LeftHandSide (a variable in the PL program). 8c3k2a

¦Expression¦ 8c3k3

The Expression is evaluated and fed to the CLI as if the user had typed it in. 8c3k3a

Note that NLSCommands may have a Result. If an NLSCommand has a Result present, the success or failure of the execution of that command is stored in the indicated BooleanLHS and is thus subsequently available to the PL program. 8c3l

```
ProcessCommand = '¦ ( Label [ParameterList] / Declaration / Statement ) ;
                                                              8c4
  Label = CH -> ':;                                           8c4a

  ParameterList = '( $<.ID ': TypeIdentifier, ',> ');         8c4b
```

ProcessCommands that have Labels can be used as objects of Process Branch commands. If they are so used and have a ParameterList, the user will be prompted to provide the values for the parameters. 8c4c

```
Declaration = Label [Persistence]
 ( TypeDec / VariableDec / ProcedureDec / RoutineDec ) ;      8c5
```

```
   Label = CH -> ':;                                                    8c5a

   Persistence = "TEMPORARY" / "SEQUENCE" / "SESSION" ;                 8c5b

   Note that Declarations may occur anywhere in a PL program.  Note also that
   because PL programs are interpreted, the scope of a Declaration name is
   determined by the execution path, not lexically.  A Declaration name is
   known and may be referenced any time between the time the interpreter has
   seen the declaration and the time the variable is destroyed.  The
   Persistence determines when the variable will be destroyed:          8c5c

      TEMPORARY variables are destroyed when a RETURN statement is executed.
                                                                        8c5c1
      SEQUENCE variables are destroyed when a FINISH statement is executed.
                                                                        8c5c2
      SESSION variables are destroyed when an AUGMENT "session" ends.
                                                                        8c5c3
      The default Persistence is TEMPORARY.                             8c5c4

   TypeDec = "TYPE" '= TypeSpecification ;                              8c5d

   VariableDec = TypeSpecification [('_/'=) Expression] ;               8c5e

   ProcedureDec = ProcedureTS '=
    $( Declaration / Statement )
    "¦END." ;                                                           8c5f

   RoutineDec = RoutineTS ;                                             8c5g

TypeSpecification =
 ArrayTS / BooleanTS / CharacterTS / CommandwordTS / EnumerationTS /
 IntegerTS / IntervalTS / ProcedureTS / RoutineTS / SelectionTS /
 TextTS ;                                                               8c6

   OrderedTS = SelectionTS / CountTS ;
   CountTS   = CharacterTS / EnumerationTS / IntgrTS ;
   IntgrTS   = IntervalTS  / IntegerTS ;                                8c6a

   StringTS = CommandWordTS / SelectionTS / TextTS ;                    8c6b

   RangeTS = IntervalTS / EnumerationTId ;                              8c6c

   ArrayTS       = ArrayTId       / ArrayTypeConstructor;
   BooleanTS     = BooleanTId;
   CharacterTS   = CharacterTId;
   CommandWordTS = CommandWordTId;
   EnumerationTS = EnumerationTId / EnumerationTypeConstructor;
   IntegerTS     = IntegerTId;
   IntervalTS    = IntervalTId    / IntervalTypeConstructor;
```

```
   ProcedureTS    = ProcedureTId    / ProcedureTypeConstructor;
   RoutineTS      = RoutineTId      / RoutineTypeConstructor;
   SelectionTS    = SelectionTId;
   TextTS         = TextTId;                                        8c6d

   ArrayTypeConstructor = "ARRAY" RangeTS "OF" TypeSpecification;
                                                                    8c6e
   EnumerationTypeConstructor = '{ #<.ID, ',> '};                   8c6f

   IntervalTypeConstructor = ('(/'[) OrderedExp ', OrderedExp (')/']) ;
                                                                    8c6g
   ProcedureTypeConstructor = "PROCEDURE" [ParameterList] [ReturnsClause];
                                                                    8c6h
      ParameterList = '( $<.ID ': TypeIdentifier, ',> ');           8c6h1

      ReturnsClause = "RETURNS" '( $<TypeIdentifier, ',> ');        8c6h2

   RoutineTypeConstructor =
    ("FEROUTINE"/"BEROUTINE") [ParameterList] [ReturnsClause];      8c6i

      ParameterList = '( $<.ID ': TypeIdentifier, ',> ');           8c6i1

      ReturnsClause = "RETURNS" '( $<TypeIdentifier, ',> ');        8c6i2

      Before a PL program can call an L10 routine in the FrontEnd or Backend,
      the name and calling sequence of that routine must be specified.  Note
      that this is analagous to CML programs.                       8c6i3

TypeIdentifier =
 ArrayTId / BooleanTId / CharacterTId / CommandWordTId / EnumerationTId /
 IntegerTId / IntervalTId / ProcedureTId / RoutineTId / SelectionTId /
 TextTId ;                                                          8c7

   OrderedTId = SelectionTId / CountTId ;
   CountTId   = CharacterTId / EnumerationTId / IntgrTId ;
   IntgrTId   = IntervalTId  / IntegerTId ;                         8c7a

   StringTId = CommandWordTId / SelectionTId / TextTId ;            8c7b

   RangeTId = EnumerationTId / IntervalTId ;                        8c7c

   ArrayTId       = .ID;
   BooleanTId     = .ID / "BOOLEAN" ;
   CharacterTId   = .ID / "CHARACTER" ;
   CommandWordTId = .ID / "COMMAND-WORD" ;
   EnumerationTId = .ID;
   IntegerTId     = .ID / "INTEGER" ;
   IntervalTId    = .ID;
   ProcedureTId   = .ID;
```

```
RoutineTId      = .ID;
SelectionTId    = .ID / "SELECTION" ;
TextTId         = .ID / "TEXT" ;                                    8c7d

Selections are meant to hold AUGMENT addresses in text form, i.e., they
should be convertable by 'caddexp', AUGMENT' address evaluation routine,
into L10 TEXT POINTERS.                                             8c7e

Note also that when Selections are taken as pointers to nodes in an AUG-
MENT file tree structure, they are strictly ordered if they point to the
same file.  Thus there can be intervals/sequences of Selections that can
be iterated over.                                                   8c7f

Statement = [Label] (
 AssignmentStmt / BlockStmt / BumpStmt / CallStmt / CaseStmt / EchoStmt /
 ExitStmt / FinishStmt / GotoStmt / IfStmt / IterativeStmt / NullStmt /
 RepeatStmt / ReturnStmt ) ;                                        8c8

  Label = CH -> ':;                                                 8c8a

  AssignmentStmt = #<LeftHandSide, ',> '_ #<Expression, ',>;        8c8b

  BlockStmt = "BEGIN" $Statement "!END";                            8c8c

  BumpStmt = "BUMP"
   ["UP"/"DOWN"] #<CountLHS, ',> /
   ["NEXT"/"BACK"/"SUCCESSOR"/"PREDECESSOR"] #<SelectionLHS, ',> ;
                                                                    8c8d
    The defaults are UP and SUCCESSOR.                              8c8d1

    Note that since Selections have order, they may be BUMPed.
                                                                    8c8d2
  CallStmt = "CALL" Call;                                           8c8e

    Call = ( RoutineExp / SelectionExp ) [ '( $<Expression, ',> ') ];
                                                                    8c8e1
    Any of FEROUTINEs, BEROUTINEs, or PL PROCEDURES may be Called.
                                                                    8c8e2
  CaseStmt = "CASE" Expression "OF"
   #StmtChoice
   "!ENDCASE" ': Statement;                                         8c8f

    StmtChoice = '! #<RelationTail, ',> ': Statement;               8c8f1

    RelationTail =
     ('=/'#) Expression /
     ('>/'</">="/"<=") OrderedExp /
     ["NOT"] "IN" RangeTS ;                                         8c8f2
```

```
   Note that since Selections have order, they may be compared/related
   using any of the comparison/relational operators.          8c8f3

EchoStmt = "ECHO" BooleanExp;                                 8c8g

   The value of the BooleanExp determines whether the user will see things
   happen in the Command Feedback Window.                     8c8g1

ExitStmt = ( "EXITBLOCK" / "EXITLOOP" / "EXITCASE" ) [IntgrExp];
                                                              8c8h
FinishStmt = "FINISH";                                        8c8i

   There is an implicit FinishStmt after the end of any PCS, whether the
   limits of the sequence was determined by the selections in a Process
   Group command or any other way.                            8c8i1

GotoStmt = "GOTO" SelectionExp;                               8c8j

IfStmt =
 "IF" BooleanExp "THEN" Statement
 ["!ELSE" Statement];                                         8c8k

IterativeStmt =
 [ Iteration / Assignation ] [ConditionTest] "DO"
    Statement
 [ConditionTest];                                             8c8l

   Iteration = [ "FOR" OrderedLHS ] "IN" RangeTS [Increment];
                                                              8c8l1
      Increment =
       ("UP"/"DOWN") [IntgrExp] /
       "USING" ("NEXT"/"BACK"/"SUCCESSOR"/"PREDECESSOR") ;    8c8l1a

      The USING can be used only if the OrderedExp in the RangeTS is of
      the Selection TYPE.                                     8c8l1b

   Assignation = "FOR" LeftHandSide '_ Expression ', Expression;
                                                              8c8l2
   ConditionTest = ("WHILE"/"UNTIL") BooleanExp;              8c8l3

NullStmt = "NULL";                                            8c8m

RepeatStmt =
 ( "REPEATBLOCK" / "REPEATLOOP" ) [IntgrExp] /
 "REPEATCASE" [IntgrExp] "WITH" Expression;                   8c8n

ReturnStmt = "RETURN" [ '( $<Expression, ',> ') ];            8c8o

   A ReturnStmt executed at that "highest" level acts as a FinishStmt.
                                                              8c8o1
```

```
LeftHandSide = LHSList /
 ArrayLHS / BooleanLHS / CharacterLHS / CommandWordLHS / EnumerationLHS /
 IntegerLHS / IntervalLHS / ProcedureLHS / RoutineLHS / SelectionLHS /
 TextLHS ;                                                          8c9

   LHSList = '( 2#<LeftHandSide, ',> ') ;                           8c9a

      This form may be used to store multiple return values from a Call.
                                                                    8c9a1
   OrderedLHS = SelectionLHS / CountLHS ;
   CountLHS   = CharacterLHS / EnumerationLHS / IntgrLHS ;
   IntgrLHS   = IntervalLHS  / IntegerLHS ;                         8c9b

   StringLHS = CommandWordLHS / SelectionLHS / TextLHS ;            8c9c

   RangeLHS = EnumerationLHS / IntervalLHS ;                        8c9d

   ArrayLHS       = .ID / IndexedReference ;
   BooleanLHS     = .ID / IndexedReference ;
   CharacterLHS   = .ID / IndexedReference ;
   CommandWordLHS = .ID / IndexedReference ;
   EnumerationLHS = .ID / IndexedReference ;
   IntegerLHS     = .ID / IndexedReference ;
   IntervalLHS    = .ID / IndexedReference ;
   ProcedureLHS   = .ID / IndexedReference ;
   RoutineLHS     = .ID / IndexedReference ;
   SelectionLHS   = .ID / IndexedReference ;
   TextLHS        = .ID / IndexedReference ;                        8c9e

   IndexedReference = ArrayLHS '[ OrderedExp '] ;                   8c9f

Expression =
 ArrayExp / BooleanExp / CharacterExp / CommandWordExp / EnumerationExp /
 IntegerExp / IntervalExp / ProcedureExp / RoutineExp / SelectionExp /
 TextExp ;                                                          8c10

   OrderedExp = SelectionExp / CountExp ;
   CountExp   = CharacterExp / EnumerationExp / IntgrExp ;
   IntgrExp   = IntervalExp  / IntegerExp ;                         8c10a

   StringExp = CommandWordExp / SelectionExp / TextExp ;            8c10b

   RangeExp = EnumerationExp / IntervalExp ;                        8c10c

   ArrayExp       = AnyTypeExp / '[ #<Expression, ',> '] ;          8c10d

   BooleanExp     = AnyTypeExp /                                    8c10e
```

```
   BooleanExp "OR"  BooleanExp /
   BooleanExp "AND" BooleanExp /
   "NOT" BooleanExp /                                          8c10e1

   Expression ('=/'#) Expression /
   OrderedExp ('>/'</">="/"<=") OrderedExp /
   OrderedExp ["NOT"] "IN" RangeTS /                           8c10e2

   "EXIST" '( SelectionExp ') /                                8c10e3

      This expression will test whether or not the specified AUGMENT ad-
      dress actually exists.                                   8c10e3a

   "TRUE" / "FALSE" ;                                          8c10e4

CharacterExp    = AnyTypeExp / ("FIRST"/"LAST") IntervalTId / .SR1 ;
                                                               8c10f
   The OrderedExp associated with the IntervalTId must be of CHARACTER
   TYPE.                                                       8c10f1

CommandWordExp = AnyTypeExp / .SR ;                            8c10g

EnumerationExp = AnyTypeExp / ("FIRST"/"LAST") RangeTId ;      8c10h

   The OrderedExp associated with the RangeTId must be of Enumeration
   TYPE.                                                       8c10h1

IntegerExp      = AnyTypeExp /                                 8c10i

   IntgrExp    ('+/'-)    IntgrExp /
   IntgrExp ('*/'//"MOD") IntgrExp /
   '- IntgrExp                                                 8c10i1

   "LEVEL" '( SelectionExp ') /                                8c10i2

      The AUGMENT "level" of a statement.                      8c10i2a

   (ArrayLHS/StringLHS/.SR/RangeTId) ".L" /
   ("FIRST"/"LAST") RangeTId /                                 8c10i3

   ("MIN"/"MAX") '( 2#<IntgrExp, ',> ') /
   "ABS" '( IntgrExp ') /                                      8c10i4

   .NUM ;                                                      8c10i5

IntervalExp     = AnyTypeExp /                                 8c10j
```

```
   IntervalExp    ('+/'-)    IntervalExp /
   IntervalExp ('*/'//"MOD") IntervalExp /
   '- IntervalExp                                              8c10j1

   "LEVEL" '( SelectionExp ') /                                8c10j2

      The AUGMENT "level" of a statement.                      8c10j2a

   (ArrayLHS/StringLHS/.SR/RangeTId) ".L" /
   ("FIRST"/"LAST") RangeTId /                                 8c10j3

   ("MIN"/"MAX") '( 2#<IntervalExp, ',> ') /
   "ABS" '( IntervalExp ') /                                   8c10j4

   .NUM ;                                                      8c10j5

ProcedureExp   = AnyTypeExp / '$ ProcedureLHS ;                8c10k

RoutineExp     = AnyTypeExp / '$ RoutineLHS ;                  8c10l

SelectionExp   = AnyTypeExp /                                  8c10m

   ("BACK"/"DOWN"/"END"/"HEAD"/"NEXT"/"ORIGIN"/
    "PREDECESSOR"/"SUCCESSOR"/"TAIL"/"UP")
      '( SelectionExp ') /                                     8c10m1

      These Expressions ennable moving around AUGMENT tree-structured
      files.                                                   8c10m1a

   .SR ;                                                       8c10m2

TextExp        = AnyTypeExp / .SR ;                            8c10n

AnyTypeExp = '( Expression ') / .ID /
 AssignmentExp / Call / CaseExp / IfExp / IndexedReference / UserExp ;
                                                               8c10o
   AssignmentExp = LeftHandSide ('_/":=") Expression ;         8c10o1

   Call = ( RoutineExp / SelectionExp ) '( $<Expression, ',> ') ;
                                                               8c10o2
      Any of FEROUTINEs, BEROUTINEs, or PL PROCEDURES may be Called.
                                                               8c10o2a
   CaseExp = "CASE" Expression "OF"
    #ExpChoice
    "¦ENDCASE" ': Expression;                                  8c10o3

      ExpChoice = '¦ #<RelationTail, ',> ': Expression;        8c10o3a

      RelationTail =
```

```
        ('=/'#) Expression /
        ('>/'</">=">/"<=") OrderedExp /
        ["NOT"] "IN" RangeTS ;                                  8c10o3b

    IfExp = "IF" BooleanExp "THEN" Expression "ELSE" Expression;
                                                                8c10o4
    IndexedReference = ArrayLHS '[ OrderedExp '] ;              8c10o5

    UserExp = '¦ (LeftHandSide '_ / TypeIdentifier) '¦;         8c10o6

        The user will be interactively prompted to provide a value.  The
        user-provided value will be evaluated according to the TYPE indi-
        cated by the LeftHandSide or TypeIdentifier.            8c10o6a

        The user will be inputting a LSEL.  Thus he or she may BUG rather
        than type.                                              8c10o6b

CNwM = Comment / NoiseWords / Message ;                          8c11

    Comment = '% -> '%;                                          8c11a

    NoiseWords = '( -> ');                                       8c11b

    Message = '; -> ';;                                          8c11c

    A CNwM may appear any place a space may appear.  Comments are ignored.
    The value of a NoiseWord will appear in the Command Feedback Window
    enclosed in parentheses.  The value of a Message will appear in the TTY
    Window enclosed in semi-colons.                              8c11d

    Both NoiseWords and Messages may be variable.  They both may contain:  '¦
    Expression '¦.  If so, the Expression is evaluated, converted to text if
    necessary, and shown in the appropriate window.              8c11e
```

Glossary 9

CML: The Command Meta Language. The user interface of all NLS subsystems is specified in CML. 9a

DAD: Do-All Debugger 9b

encapsulation: There is a facility for encapsulating programs for NLS. This facility allows programs (.sav or .exe files) that were written without NLS in mind to be executed as sub-forks of NLS. Encapsulated programs may get their input from NLS files or, to a degree, interactively from the user. Similarly for the output from encapsulated programs. Currently the Meta, L10, and CML compilers are encapsulated. 9c

fork: TENEX's term for what is most often called a "process" in computer terminology 9d

INCLUDE statement: We have facility for "including" a group of statements from any NLS file as if that group of statements were actually present in place of the INCLUDE statement. 9e

index, a LIBRARY: An index file is produced by the LIBRARY subsystem from a source code file. It contains a sorted list of all the global variable and procedure names in that module with pointers to their locations in the source code file. See "SysGuide". 9f

JDAD: JOVIAL DAD 9g

L10: An ALGOL-like language with additional string manipulation facilities. L10 is the primary implementation language of NLS. 9h

LIBRARY subsystem: An NLS subsystem that will conditionally perform various clerical and bookkeeping chores on a collection of modules, e.g., compiling, loading, printing, indexing, contructing SysGuides. The reference manual may be found in <29151,>. 9i

Meta: A "meta-compiler" system used to produce compilers. L10, CML, and of course Meta are written in Meta. 9j

PROGRAMS subsystem: An NLS subsystem having commands of use to programmers, e.g., Compile, Insert Procedure. Users' documentation may be found in <ArcDocumentation,Programs,>. 9k

subsystem, an NLS: NLS may be viewed as a collection of subsystems. Each subsystem has a collection of commands that are functionally related, e.g., the BASE subsytem has editing commands (and some others). 9l

SysGuide: A sorted collection of indices (see "index" above). Typically a SysGuide will contain the indices from all the modules in the entire scope of an address space (in a fork or .sav file). A SysGuide may be used in the NLS Jump (to) Name External command. 9m

template: See <SsSrc,Programs-Templates,>. A group of NLS statements used by the PROGRAMS subsystem Insert command, e.g., 9n

```
UNTIL until-clause DO                    9n1
   BEGIN                                 9n1a
   END;                                  9n1b
```

templates file: An NLS file containing templates, see <SsSrc,Programs-Templates,>. 9o

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.